National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

Adaptive Subdivision of Bézier Curves and Surfaces

by

Chihei Cheong

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall 1992

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-77062-7

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Chihei Cheong

TITLE OF THESIS: Adaptive Subdivision of Bézier Curves and Surfaces

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1992

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) Cheong C.L. H.

Permanent Address:
Apt 706,
5 Greystone Walk Drive,
Scarborough, Ontario,
Canada, M1K 5J5

Date: Sept. 3. 92

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Adaptive Subdivision of Bézier Curves and Surfaces** submitted by **Chihei Cheong** in partial fulfilment of the requirements for the degree of Master of Science.

Barry Joe
.....
Supervisor: Barry Joe

.....
Committee member: Mark Green

.....
Committee member: Yau Shu Wong

Date: Sept. 1, 92

Abstract

Linear approximation of Bézier curves and surfaces is a common pre-processing step for many algorithms because many algorithms can only operate efficiently on linear primitives. Adaptive subdivision is a procedure that adaptively subdivides a curve or surface such that each curve segment or surface patch satisfies a flat tolerance and can be approximated by a linear primitive. The advantage of adaptive subdivision is that the flat regions of the Bézier curves or surfaces will be approximated with fewer primitives and the curved regions will be approximated by more primitives. Different adaptive subdivision algorithms have been developed before. This thesis presents a new adaptive Bézier curve subdivision method that utilizes the curvature and arc-length information of the curve to subdivide a curve. Our empirical results show that the new method can approximate a curve with fewer linear segments than the well known recursive adaptive subdivision within the same flat tolerance. This new method is also used for improving the traditional recursive Bézier triangle subdivision algorithm. The recursive algorithm always subdivides the triangle's boundary at the 0.5 parameter point. Our new curve subdivision method chooses the optimal splitting positions on the triangle's boundaries. As a result, the surface can be approximated with fewer planar triangles than the traditional method within the same flat tolerance. We also design a new scheme for choosing the triangle subdivision patterns based on the shape and flatness of the triangle's boundaries. The new scheme is aimed at reducing long and thin triangles that are occasionally generated during the subdivision procedure.

Acknowledgements

I would like to thank my supervisor, Dr. Barry Joe for the time, patience, and valuable insights he provided during the development of this thesis. I would also like to thank the members of my examining committee, Dr. Yau Shu Wong and Dr. Mark Green for their time and effort in reviewing this research, and Dr. Joe Culberson for chairing the committee.

Special thanks to Wenping Wang and Anwei Lui for the discussions on subdivision problems.

Finally, I would like to thank Dora Wong for her patience and support throughout this work and my parents who gave me the encouragement and opportunity to pursue my dreams.

Table of Contents

1 Introduction 1

2 Bézier Curves and Surfaces 3
- 2.1 Definitions of Bézier Curves and Surfaces 3
- 2.2 Linear Approximation of Bézier Curves and Surfaces 5
- 2.3 Criteria of Flatness 10

3 Bézier Curve Subdivision by Curvature Arc-length Graph 18
- 3.1 Background on Recursive Adaptive Subdivision 18
- 3.2 Subdivision by Curvature Arc-length Graph 20
  - 3.2.1 Example of Subdividing a Bézier Curve 22
- 3.3 Relation between Chord Distance and Area under "Curvature Arc-length" Graph 23
  - 3.3.1 Chord Distance of a Circular Arc Sequence 25
  - 3.3.2 Arcs Reduction 27
  - 3.3.3 Relative Chord Distances 29
  - 3.3.4 Local Minimum and Local Maximum in Curvature Arc-length Graph 33
- 3.4 Empirical Results 40
- 3.5 Conclusion 47

4 Bézier Triangle Subdivision 48
- 4.1 Background 50
- 4.2 Bézier Triangle Subdivision 56
  - 4.2.1 Avoidance of Small Peaks on the Curvature Arc-length Graph 66
  - 4.2.2 Example of Bézier Triangle Subdivision 67
- 4.3 Empirical Results 70
- 4.4 Conclusion 77

5 Concluding Remarks 79
- 5.1 Future Improvements on Curve Subdivision Algorithm 80
  - 5.1.1 Adaptive Bounding Box 80
- 5.2 2-Dimension Extension of Curvature Arc-length Graph 81

Bibliography 83

List of Tables

2.1 Comparison of Gopalsamy's method and convex hull method for determining upper bounds on Rational Bézier curves 17

2.2 Comparison of Gopalsamy's method and convex hull method for determining upper bounds on Non-Rational Bézier curves 17

3.1 $0.5(8e)^{1/2} <$ Sectional area $A_{si} <= (8e)^{1/2}$ 40

3.2 $0.5(8e)^{1/2} <$ Sectional area $A_{si} <= 0.95(8e)^{1/2}$ 45

3.3 $0.5(8e)^{1/2} <$ Sectional area $A_{si} <= 0.90(8e)^{1/2}$ 46

4.1a Rational Bézier Triangle Subdivision using patterns (c,a,d) and (c,c,c) .. 53

4.1b Comparison of maximum angles 54

4.1c Comparison of minimum angles 54

4.2a Rational Bézier Triangle Subdivision using our method and Filip's method 70

4.2b Comparison of maximum angles 71

4.2c Comparison of minimum angles 72

List of Figures

2.1	Chordal distance of a Bézier curve	13
2.2	Chordal distance less than the maximum deviation between the approximating line and the Bézier curve	13
2.3	Convex hull method to measure upper bound of chordal distance	14
3.1a	Example on subdividing a Bézier curve	22
3.1b	Curvature arc-length graph of Figure 3.1a	23
3.2a,b	A single arc and Curvature Arc-length graph	23
3.3	Arc sequence M_i, m=4, i=0	28
3.4a,b	Arc sequences M_i and M_{i+1} for case 2 reduction	30
3.5	Proof of $\sin(a_1) / \sin(a_2) < a_1/a_2$	32
3.6a,b	Arc sequences M_i and M_{i+1} for case 3 reduction	32
3.7a,b	Arc sequence M_0	35
3.8a,b	Arc sequence M_1	35
3.9a,b	Arc sequence M_2	35
3.10a,b	Arc sequence M_3	36
3.11a,b	Arc sequence M_0	37
3.12a,b	Arc sequence M_1	37
3.13a,b	Arc sequence M_2	38
3.14a,b	Arc sequence M_3	38
4.1	Four patterns for subdividing a Bézier triangle	50
4.2	A Bézier triangle subdivision example from [Fili86a], using pattern combinations (c,c,c), (c,a,d) and (b,a,d) respectively.	52
4.3	Wide, narrow, and normal triangles	57
4.4	Rule 2.2.1 for a and b, Rule 3.2.2 for c.	58
4.5	Pattern 'b' subdivision	59
4.6	Bounding box on the "curvature arc-length" graph	67
4.7	Bézier triangle subdivision by our method	68
4.8	Bézier triangle subdivision by Filip's method, using pattern (c,a,d)	68
5.1	Adaptive bounding boxes on "curvature arc-length" Graph	80

Chapter 1

Introduction

Bézier curves and surfaces are very well-established design tools in computer aided geometric design. However, many algorithms for display, rendering, and finite element analysis can only operate efficiently on linear primitives such as lines and polygons. In order to apply these algorithms on the Bézier curves and surfaces, the curves and surfaces have to be first approximated by polylines and polygon meshes. The procedure of approximating Bézier curve and surface with linear primitives is also known as the refinement procedure.

The refinement procedure is a pre-processing step for many algorithms that cannot directly operate on Bézier curves and surfaces. The performance of different refinement procedures can be measured by the execution time or by the number of output primitives. Different applications prefer different trade-offs between execution time and number of output primitives. Some applications, such as interactive graphics display, need a fast refinement procedure to supply the linear primitives to the display hardware. Other applications, such as raytracing, prefer a refinement procedure that generates fewer primitives to approximate a complex object. Still other applications, such as analyzing the stress distribution on an airplane body, only approximate the curved surface once and use the approximating primitives repeatedly over many times. For the second and third applications, a refinement procedure that generates fewer number of primitives within a given tolerance is very important to ensure efficient performance. Our interest is to find

a procedure that can approximate a Bézier curve or surface with fewer linear primitives than other currently used procedures within a given tolerance. Since recursive adaptive subdivision proposed by [Lane80, Bars87, Fili86a] is one of the best methods to generate the fewest number of approximating primitives, our objective is to design a procedure that generates fewer number of primitives than the recursive method.

In Chapter 2, we give the definitions of Bézier curves and surfaces. We will discuss other refinement procedures and their trade-off between execution speed and the number of output primitives. We will then compare different criteria on flat tolerance of curve segments and surface patches.

In Chapter 3, we present our subdivision algorithm on Bézier curves. We first describe the flatness criterion that we use. Then we present our refinement procedure and give an explicit example. We then explain the motivation of our algorithm, such as the motivation for splitting at the local maximum of the "curvature arc-length" graph. Finally, we compare the number of output segments of our method with that of the recursive adaptive subdivision.

In Chapter 4, we apply our curve refinement procedure to improve the recursive Bézier triangle subdivision algorithm proposed by [Fili86a]. We first give the background on Bézier triangle subdivision and discuss how to compare the number of output triangles. Then we describe how our curve subdivision algorithm can improve the triangle subdivision. Finally, we compare the performance of the improved method and the original recursive subdivision.

In Chapter 5, we present the conclusions and discuss some future considerations.

Chapter 2

Bézier Curves and Surfaces

The linear approximation of Bézier curves and surfaces is also known as the refinement procedure. The design of a refinement procedure can be based on two main objectives: the execution speed and the number of output polylines or polygons. A fast refinement procedure may approximate a curve or surface in real time while generating large number of polylines or polygons. A slow procedure may take longer to approximate a curve or surface but generate fewer number of elements. In Section 2.2, we will discuss different algorithms that achieve either aspect of the objectives.

Each refinement procedure may choose its own criterion on flatness. A flatness criterion determines when a curve segment or a surface patch is flat enough to be approximated by a line or a polygon. It determines when the refinement procedure may stop. In Section 2.3, we will discuss different kinds of flatness criteria.

First of all, we will give the definitions of Bézier curves and surfaces in Section 2.1.

2.1 Definitions of Bézier Curves and Surfaces

A Bézier curve of degree n is defined by a control polygon of (n+1) vertices. Let the control vertices be $\{\mathbf{P}_j \mid j=0,..,n\}$ and the weights be $\{w_j>0. \mid j=0,..,n\}$. A Bézier curve is defined as

$$\mathbf{b}(u) = \frac{\sum_{j=0}^{n} B_j^n(u)\, w_j\, \mathbf{P}_j}{\sum_{j=0}^{n} B_j^n(u)\, w_j}$$

where $u \in [0,1]$

$$B_j^n(u) = \binom{n}{j} u^j (1-u)^{n-j}$$

The $B^n_j(u)$, j=0,..,n, are known as the univariate Bernstein polynomials. For non-rational Bézier curves, all weights are equal to 1 and the denominator becomes 1.

There is more than one form of Bézier surface. Two of the most common forms are the Bézier rectangular tensor product patch and the Bézier triangle. A Bézier rectangle of degree m×n is defined over a rectangular control graph of (m+1)×(n+1) vertices. Let the control vertices be $\{\mathbf{P}_{i,j} \mid i=0,..,m,\ j=0,..,n\}$ and the weights be $\{w_{i,j}>0. \mid i=0,..,m,\ j=0,..,n\}$. A Bézier rectangle is defined as follows:

$$\mathbf{b}(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)\, B_j^n(v)\, w_{i,j} P_{i,j}}{\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)\, B_j^n(v)\, w_{i,j}}$$

where $u, v \in [0,1]$

For non-rational Bézier rectangles, all weights are equal to 1 and the denominator becomes 1.

A Bézier triangle of degree n is defined over a triangular control graph of (n+1)(n+2)/2 control vertices. Let the control vertices be $\{\mathbf{P}_{i,j} \mid i+j=n,\ i>=0,\ j>=0\}$ and the weights be $\{w_{i,j}>0. \mid i+j=n,\ i>=0,\ j>=0\}$. A Bézier triangle is defined as follows:

$$\mathbf{b}(u,v) = \frac{\sum_{i+j=0}^{n} B_{i,j}^n(u,v)\, w_{i,j}\, \mathbf{P}_{i,j}}{\sum_{i+j=0}^{n} B_{i,j}^n(u,v)\, w_{i,j}}$$

where $u, v \in [0,1]$

$$B_{i,j}^n(u,v) = \binom{n}{i,j} u^i v^j (1-u-v)^{n-i-j}$$

$$\binom{n}{i,j} = \frac{n!}{i!\, j!\, (n-i-j)!}$$

The $B^n_{i,j}(u,v)$ are known as the bivariate Bernstein polynomials. For non-rational Bézier triangles, all weights are equal to 1 and the denominator becomes 1.

Bézier curves and surfaces possess many nice properties [Fari90]. Two of these are the interpolation of corner control vertices and the convex hull property: the curve and surface lie in the convex hull of the control vertices. The convex hull property will be used to determine the flatness of Bézier curves and surfaces. Details will be discussed in sections 2.2 and 2.3.

[Gold87] gives an explicit formula for converting a m×n degree Bézier rectangle to two Bézier triangles of degree (m+n). Any algorithm that works on Bézier triangles can be applied to Bézier rectangles by converting the rectangles to two triangles. For this reason, an algorithm designed for Bézier triangles is more versatile than an algorithm designed for Bézier rectangles. Therefore, we design our surface refinement algorithm based on the Bézier triangular surface.

However, the Bézier rectangle has been commonly known for a longer period of time than the Bézier triangle. Most of the refinement algorithms are first developed on Bézier rectangles. Although our proposed surface refinement algorithm is for Bézier triangles, we also discuss other algorithms designed for Bézier rectangles in Sections 2.2 and 2.3.

2.2 Linear Approximation of Bézier Curves and Surfaces

One of the fastest refinement methods for Bézier curves is the forward differencing method [Bart87]. This method samples a Bézier curve along equally spaced

parameter values and approximates each curve segment between two adjacent samples with a line segment. The method pre-computes the incremental values for the curve polynomial functions and derives each sample point by adding the incremental values to the previous sample point. If the Bézier curve is a 3-D non-rational curve of degree n, each sample requires only (3n) addition operations. If the Bézier curve is a 3-D rational curve of degree n, each sample point requires (4n) addition and 3 division operations.

Forward differencing is a very fast method because each sample requires only very few arithmetic operations. However, this method will not sample the curve adaptively because it samples with a fixed parameter step size. In other words, a fixed step size will not sample a curve more on high curvature regions and less on low curvature regions. Therefore, forward differencing is not a perfect solution for generating the fewest number of approximating primitives.

An attempt to solve the fixed step size on the cubic curves is proposed by [Lien87]. This method is called adaptive forward differencing. This method will adjust the step size by reparameterizing the cubic curve after each sample. The reparameterization will double or half the original parameter step size and as a result, sample more or less densely accordingly. The control of doubling and halving the step size is determined by the arc length of the previous sample. However, their proposed method is only for cubic curves and always samples the curve at pixel distance. In other words, their method is a screen co-ordinate algorithm that generates line segments at pixel co-ordinates. A screen co-ordinate algorithm is not suitable for other applications that require world co-ordinate linear approximations.

One solution to generate fewer approximating primitives than forward differencing is the recursive adaptive subdivision algorithm [Lane80a, Bars87, Fili86a]. The recursive algorithm for Bézier curves recursively subdivides a Bézier curve until each curve segment is flat enough to be approximated by a line segment. The algorithm usually subdivides a curve at the mid-point parameter into two Bézier curve segments. Each segment is subjected to a flatness test. If a curve segment is determined to be flat, it is approximated by a line segment joining the two end points of the curve segment. If a curve segment is determined to be non-flat, it is recursively subdivided until all the subdivided curve segments are considered to flat.

The recursive algorithm for Bézier surfaces recursively subdivides a Bézier surfaces until each subdivided patch is flat enough to be approximated by a polygon. The algorithm usually subdivides the boundaries of the surface (a Bézier rectangle or Bézier triangle) at the mid-point parameter and splits the surface into two or more Bézier surfaces. Each Bézier surface is subjected to a flatness test. If a Bézier surface is determined to be flat, it is approximated by a polygon (a triangle or quadrilateral) formed by the corners of the Bézier surface. If a Bézier surface is determined to be non-flat, it is recursively subdivided until all the subdivided Bézier patches are considered to be flat .

The recursive adaptive subdivision approximates a curve or surface by more segments at curved regions and fewer segments at flat regions. Algorithms proposed by [Lane80a, Bars87, Fili86a] use a common flatness test that measures the perpendicular distance between the Bézier control vertices and the approximating primitives. Their flatness test , referred to as the convex hull method, will be discussed in section 2.3. Both Lane's and Barsky's algorithms subdivide Bézier rectangular surfaces. Barsky

proposes an adaptive flat tolerance that will change according to the distance between the object and the eye position. If the object is far from the eye point, the object is subjected to a large flat tolerance. If the object is close to the eye point, the object is subjected to a small flat tolerance. The adaptive flat tolerance will reduce the number of approximating primitives for the objects that will appear small on the screen. Filip's algorithm is primarily for the Bézier triangle subdivision. Filip proposes a more efficient flatness testing procedure based on the pseudo-tolerance. The pseudo-tolerance can avoid some of the square-root operations during the flatness test.

There are three reasons why a recursive curve subdivision algorithm is slower than the forward differencing method. The first reason is that a recursive algorithm always needs to generate the control graphs of the subdivided curves. A control graph is generated by the de Casteljau algorithm [Fari90], which is a slower algorithm than the incremental polynomial evaluation in forward differencing. The second reason is that each subdivided Bézier segment must be subjected to a flatness test. [Bars87] and [Fili86a] empirically find that the flatness test can take as much computation time as the de Casteljau algorithm. Forward differencing uses the methods in [Fili86b] and [Chen92] to determine the parameter step size, which will guarantee the flatness of all curve segments. Therefore, the expensive flatness test is not required. The third reason is that a recursive algorithm is difficult to implement on hardware because it requires high speed stack memory [Lien87]. The size of the stack cannot be determined until execution time. On the contrary, forward differencing can be easily implemented in hardware because it only involves simple arithmetic operations.

Although the recursive adaptive algorithm is not as fast as the forward differencing method, it generates far fewer approximating lines than the latter method for the same flat tolerance. [Vasi90] also emphasizes the importance of generating fewer primitives in the refinement procedures. He proposes a non-recursive adaptive algorithm that, he claims, will use the least possible polygons to approximate a Bézier rectangular surface. His algorithm first samples a Bézier rectangle surface in the u and v parameter directions. The samples in the u direction are grouped together to form surface strips according to some flatness criteria. Then the samples in the v direction of each strip is grouped together to form surface patch according to the same flatness criteria. The output surface patches are the patches that satisfy the flatness criteria in both u and v parameter directions.

Due to the large number of samples and repeated flatness test on each sample, Vlassopoulos' algorithm is even more expensive than the recursive algorithm. The extra computation of this refinement algorithm is paid off when fewer number of output polygons are fed into the next stage of his highly expensive operation, raytracing. [Vali90] shows that some applications are concerned with the number of approximating polygons more than the computation time of the refinement procedure.

Although [Vasi90] claims that his algorithm generates the least possible number of polygons to approximate the surface, he does not compare his results with other adaptive subdivision methods. Moreover, his flatness testing method on chordal distance is a rough estimation of the actual chordal distance. His chordal distance estimation may be enough for proper visual effects in rendering, but may not be enough for other intense

applications in finite element analysis. We will further discuss his flatness test in section 2.3.

2.3 Criteria of Flatness

A flatness criterion is a rule to determine when a curve segment or surface patch is flat enough to be approximated by a line segment or polygon. Different flatness criteria are proposed based on their ease of computation and their accuracy in measuring the flatness of a curve or a surface. An accurate flatness criterion should bound the maximum deviation between the approximating primitives and the Bézier entities. The purpose of bounding the deviation is important for analyzing the possible error between the approximating primitives and the Bézier entities.

A flatness criterion can be expressed in terms of a flatness test. A flatness test is commonly used in the recursive adaptive subdivision [Lane80, Bars87, Fili86a]. Each Bézier entity is subjected to a flatness test. If it fails the test, it will be recursively subdivided until all its children pass the test. A flatness criterion can also be expressed in terms of a parameter step size or a subdivision level in a refinement process.. [Fili86b] and [Chen90] use the bounds on derivatives of a Bézier entity to determine the parameter step size. Their methods determine a step size or subdivision level that will bound the maximum deviated distance between the subdivided entities and the approximating primitives.

There are three mainly kinds of flatness criteria: criteria based on arc-length or surface area, criteria based on curvature and torsion, and criteria based on chordal

distance. An arc-length criterion is primarily for Bézier curve subdivision. This criterion restricts each subdivided curve segment to be within a pre-defined length. Since the length of a curve segment is usually difficult to find, the length between the two end points of the curve is commonly used instead. One example that uses this criterion is the adaptive forward differencing method proposed by [Lien87]. Their method uses the arc-length of previously sampled points to adjust the future sampling rate. If the current sampling distance is too short, then the next sampling parameter will be doubled. If the current sampling distance is too long, then the current sampling parameter will be halved.

Since the refinement procedure in [Lien87] compares the sampling distance with the size of a pixel, their method is quite accurate in determining the flatness of a curve segment. However, the length of a curve generally does not tell the deviated distance between the curve and the approximating line. Given two curves with equal curve length, one can be nearly flat and the other may form a loop. If two curves have equal curve length and equal length in their approximating lines through their end points, the curve containing an inflexion point will have a smaller deviation than the curve without an inflexion point [Bars87]. Therefore, the arc-length criterion is usually not accurate enough to determine the flatness of a curve.

The 2-dimensional version of an arc-length criterion is the surface-area criterion. This criterion considers any surface patch with an area smaller than a pre-defined tolerance as a flat surface. Again, this criterion is not an accurate model for testing surface flatness. The argument is similar to the curve case discussed above.

The second flatness criterion is the curvature criterion. For a given Bézier curve, this criterion measures the angle between the tangents at two ends of the curve as an

indicator of the curve flatness. If the angle is larger than a pre-defined angle, then the curve is considered not flat. A variation of the curvature criterion is the torsion criterion, which is discussed in [Patt89]. The torsion criterion uses the angle between the two binormal vectors at the end of the curve as an indicator of the curve flatness.

The 2-dimensional version of a curvature criterion measures the angle between two normals on the surface patch. Since there are infinite pairs of normals on a surface patch, [Vasi90] simplifies the test by only testing a finite number of samples in u and v parameter directions. If all pairs of normals turn less than a pre-defined tolerance angle, then the patch passes the curvature criterion.

The curvature criterion alone cannot bound the maximum deviated distance between the curve and the approximating line segment. A curve can satisfy the curvature criterion even if the deviation between the curve and the approximating line segment is large. This will happen when the curve is very long and has a very small curvature. Both [Patt89] and [Vasi90] notice this problem and both use an additional criterion to control flatness. [Patt89] proposes a simple upper bound limit on the parameter step size. [Vasi90] uses a second flatness test on chordal distance, which will be discussed next.

The third criterion is the chordal distance criterion. For a Bézier curve, the chordal distance is the maximum perpendicular distance between the curve and the line passing through the two end points (Figure 2.1). For a Bézier surface, this criterion measures the maximum perpendicular distance between the surface patch and a plane formed by three corners of the patch.

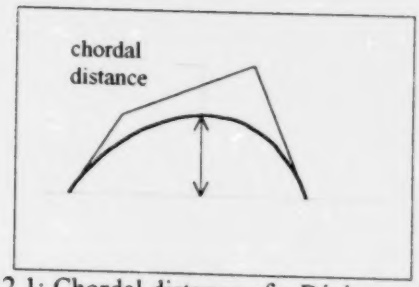


Figure 2.1: Chordal distance of a Bézier curve

Chordal distance is the most common flatness criteria used by recursive adaptive subdivision. Chordal distance usually measures the maximum deviation between the Bézier entity and the approximating primitives. There is a special case where the chordal distance does not measure the maximum deviation. The case arises when the interior control vertices of the Bézier entity are close to the approximating line or plane but far away from the control vertices at the ends or corners of the control graph. Figure 2.2 is an example of such a Bézier curve. However, this special case seldom arise in practice on the subdivided curve segments or surface patches. Therefore, chordal distance is still a very accurate criterion of measuring the flatness of a Bézier entity.

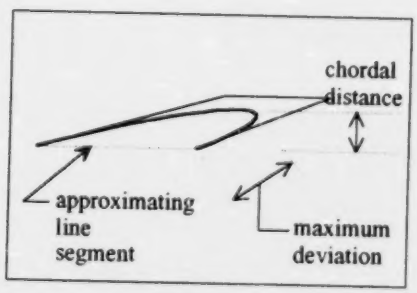


Figure 2.2: Chordal distance less than the maximum deviation between the approximating line and the Bézier curve

The exact chordal distance is usually difficult to find. [Vasi90] estimates the chordal distance by the distance between the mid-point parameter sample on the curve and the mid-point of the approximating line segment. This is a very rough estimation and is only accurate enough for visual effect in rendering.

A commonly used method to measure the upper bound of the chordal distance is referred to as the convex hull method. For a Bézier curve, this method measures the maximum perpendicular distance between the control vertices to the approximating line. The approximating line is a straight line joining two end control vertices [Figure 2.3]. Since a Bézier curve lies within the convex hull of its control graphs, this estimate always gives an upper bound of the chordal distance. The convex hull method for a Bézier surface is to measure the maximum perpendicular distance between the control vertices and the plane formed by the vertices at the three corners of the Bézier surface. The convex hull method is used by the recursive subdivision algorithms proposed by [Lane80, Bars87, Fili86a].

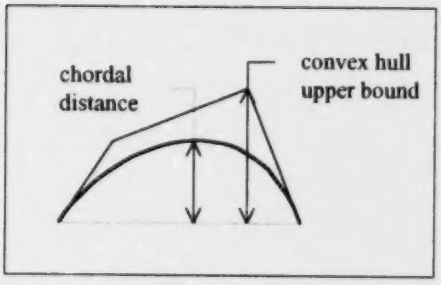


Figure 2.3: Convex hull method to measure upper bound of chordal distance

The convex hull method always over-estimates the chordal distance. A refinement algorithm that uses the convex hull method always over-subdivides a curve.

[Gopa91] proposes a new method for evaluating a more compact bound on the chordal distance. Empirically they show that their method gives a more compact bound than the convex hull method on non-rational Bézier curves. However, they do not give any performance statistics on rational Bézier curves. So we test out some random samples on rational curves as shown in Table 2.1. We use the following inequality on p.349 of [Gopa91] to find the upper bound of chordal distance of a Bézier curve:

$$|F(u)|^2 \le \frac{1}{m^2} \sum_{i=0}^{n} \left[\sum_{j=1}^{3} F_j^2(u_i) \right]$$

where $F(u) = \left(\frac{F_1(u)}{h(u)}, \frac{F_2(u)}{h(u)}, \frac{F_3(u)}{h(u)} \right)$

$|F(u)|$ = chordal distance of the Bézier curve

$F_1(u)$, $F_2(u)$, $F_3(u)$ are the polynomial functions of the chordal distance in x, y, z dimensions respectively.

m = max(lower bound of h(u) from Theorem 2 in [Gopa91],

min (w_0, .., w_n))

$h(u) = \Sigma_{j=0}^{n} B_j^n(u) w_j$

u_i = the ith golden value of degree n given in Table 1 of [Gopa91].

The format of the Table 2.1 is very similar to Table 2 in [Gopa90]. U_B is the upper bound of the chordal distance measured by convex hull method. U_L is the upper bound measured by the method suggested in [Gopa90]. Each horizontal row is the average result of 100 random Bézier curves. Each curve lies within the co-ordinate range of [0,30]×[0,30]×[0,30]. The weight on each control vertex is between 0 and 10. Columns 2 and 3 in Table 2.1 are the minimum and maximum ratios of U_B/U_L. Columns 4 and 5 are the minimum and maximum ratios of U_L/U_B. The smaller numbers in

columns 2 and 3 in comparison with columns 4 and 5 indicate that the convex hull method gives a closer bound than [Gopa90]. Column 6 shows the percentage of cases that the convex hull method gives a closer bound than [Gopa90]. The results in Table 2.1 show that the method in [Gopa90] does not give a more compact bound than convex hull method for rational Bézier curves. This is contrary to their claim on non-rational Bézier curves.

In order to show that we have the same implementation as [Gopa90], we run some non-rational Bézier curves in our program and generate the results in Table 2.2. Tables 2.1 and 2.2 use the same set of random Bézier curves. Table 2.2 shows that our results are very similar to those in [Gopa90]. This verifies the correctness of our implementation.

Since the method in [Gopa90] performs poorly for rational Bézier curves, the convex hull method is still one of the best methods to find the upper bound on chordal distance of Bézier curves.

Table 2.1: Comparison of Gopalsamy's method and convex hull method for determining upper bounds on Rational Bézier curves.

degree	ratio U_B/U_L		ratio U_L/U_B		percentage favourable to convex hull
	min	max	min	max	
2	0.0068	21.7714	0.0459	147.4943	47.0%
3	0.0111	7.7087	0.1297	90.0150	54.0%
4	0.0041	1.7927	0.5578	241.7012	82.0%
5	0.0042	2.0246	0.4939	240.6207	95.0%
1	2	3	4	5	6

Table 2.2: Comparison of Gopalsamy's method and convex hull method for determining upper bounds on Non-rational Bézier curves.

degree	ratio U_B/U_L		ratio U_L/U_B		percentage favourable to convex hull
	min	max	min	max	
2	2.0000	2.0000	0.5000	0.5000	0.0%
3	1.1880	2.8762	0.3477	0.8418	0.0%
4	0.9532	4.1830	0.2391	1.0491	3.0%
5	0.7685	3.0022	0.3331	1.3013	14.0%
1	2	3	4	5	6

Chapter 3

Bézier Curve Subdivision by Curvature Arc-length Graph

In this chapter, we present a non-recursive Bézier curve subdivision method. Our method subdivides a Bézier curve by dividing the "curvature arc-length" graph into equal area sections. A "curvature arc-length" graph plots the square-root of the curvatures against the cumulative arc length of a Bézier curve. The main advantage of our method is that it generates fewer segments than the common recursive adaptive curve subdivision algorithm [Lane80, Bars87, Fili86a].

This chapter is organised as follows: Section 3.1 discusses the background on recursive adaptive subdivisions, with which we will compare our results; Section 3.2 describes our subdivision algorithm; Section 3.3 discusses the motivation behind the design of our subdivision algorithm; Section 3.4 presents the empirical results; and finally, Section 3.5 concludes this chapter.

3.1 Background on Recursive Adaptive Subdivision

Recursive adaptive subdivision [Lane80, Bars87, Fili86a] is one of the best methods that generates the fewest number of line segments for a given flat tolerance, e. A recursive subdivision algorithm first cuts a Bézier curve at the mid-point parameter. Each one of the two sub-curves is then checked for flatness. If the sub-curve fails the

flatness test, it is recursively subdivided until all the newly generated sub-curves pass the flatness test.

The flatness of a curve segment is determined by the chordal distance between the curve segment and the approximating line. The approximating line joins the two end-points of a curve segment. The chordal distance is the maximum distance between the curve and the line, measured perpendicularly to the line.

A common way to check the upper bound of the chordal distance is to measure the maximum perpendicular distance from the Bézier control polygon to the approximating line [Bars87, Lane80, Fili86a]. If the control polygon is defined by the points $\{P_i \mid i=0,..,n\}$, then the maximum perpendicular distance between $\{P_i \mid i=1,..,n-1\}$ and the approximating line P_0P_n is the upper bound on the chordal distance. We will refer to this flatness test as the convex hull method.

The flatness test by convex hull method always over-estimates the chordal distance. A recursive subdivision algorithm that uses this flatness test usually generates more segments than necessary. Another flatness test can be used to determine the lower bound of the chordal distance. We refer to this second test as direct sampling method. The direct sampling method takes 10 equally spaced parameter samples on the curve. The maximum perpendicular distance between the samples and the approximating line segment is taken as the lower bound of the chordal distance. A recursive subdivision algorithm that uses the direct sampling flatness test will generate fewer segments than any other recursive algorithm.

We will compare the number of output segments generated by our method, with the recursive algorithm using direct sampling flatness test, and with the recursive

algorithm using convex hull flatness test. In Section 3.4, we will show that our algorithm generates, on average, fewer segments than either of the methods.

3.2 Subdivision by Curvature Arc-length Graph

In this section, we give the outline of our non-recursive subdivision algorithm. Our subdivision procedure includes 3 steps.

1. **Sample the Bézier curve.**

   The purpose of sampling is to capture the curvature and arc length information on the Bézier curve. The curve and arc length information is important for us to determine the subdivision points in step 2.

   Sampling can be done by the forward differencing method. Any standard text on curve modelling, such as [Bart87, Fari90], discusses the details. The number of equally spaced parameters can be determined by the method in [Chen92]: which uses the flat tolerance and the bounds on derivatives of the Bézier curve to determine the number of samples. Empirically, we find that this method provides adequate samples to capture the curvature and arc length information of a Bézier curve.

   The curvature at a sample point can be approximated by the curvature of a circle through the 3 consecutive sample points. [Gold90] discusses the details of finding the circle through three points. The curvature of the first and last sample on the Bézier curve can be evaluated directly from the control polygon of a

Bézier curve. The arc-length is approximated by measuring the distance between two adjacent samples.

2. Construct the "curvature arc-length" graph.

A "curvature arc-length" graph is an x-y graph where the x-axis is the cumulative arc length of the Bézier curve, and the y-axis is the square-root of the curvature of the Bézier curve. The purpose of this graph is to measure the area under the graph.

2.1. Find the area between two local maxima.

After the "curvature arc-length" graph is constructed, the graph is divided into n+1 regions, where n is the number of local maxima in the graph. We refer to a local maximum as a peak. After the graph is divided into n+1 regions, the area of each region is measured. Let the area of region i be A_i.

2.2. Divide each region into equal area sections.

If $A_i > (8e)^{1/2}$, where e is the flat tolerance, then we subdivide region i into equal area sections such that the sectional area is less than $(8e)^{1/2}$. Let A_{si} be the sectional area within the region i, then

$$A_{si} = \frac{A_i}{\lceil A_i/(8e)^{1/2} \rceil} \qquad (3.1)$$

If $A_i > (8e)^{1/2}$, then $0.5(8e)^{1/2} < A_{si} <= (8e)^{1/2}$ (3.2)

We will explain how the value $(8e)^{1/2}$ is derived in Section 3.3.

3. **Obtain the Bézier curve subdivision points**

After the graph is subdivided at the peaks and into equal area sections, we obtain the Bézier curve subdivision points in terms of the arc length positions. The subdivision points can be converted to the original parameter of the Bézier curve by linear interpolation. The conversion is the reverse process of measuring the cumulative arc length from the sample points in step 1.

We will give an example to illustrate our curve subdivision method before we discuss the motivation behind the "curvature arc-length" graph in Section 3.3.

3.2.1 Example of Subdividing a Bézier Curve

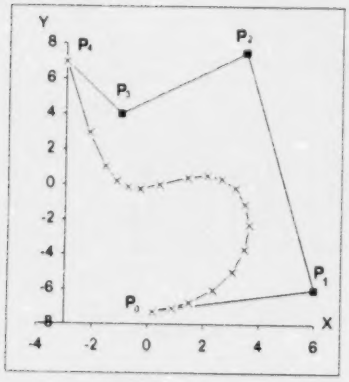


Figure 3.1a

$\mathbf{P}_0 = (P_{x0}, P_{y0}, P_{z0}) = (0.2, -7.25, 6.75)$
$\mathbf{P}_1 = (P_{x1}, P_{y1}, P_{z1}) = (6., -6., 5.)$
$\mathbf{P}_2 = (P_{x2}, P_{y2}, P_{z2}) = (3.5, 7.5, 3.5)$
$\mathbf{P}_3 = (P_{x3}, P_{y3}, P_{z3}) = (-1., 4., -3.5)$
$\mathbf{P}_4 = (P_{x4}, P_{y4}, P_{z4}) = (-3., 7., 0.)$
$w_0 = 1.$ $w_1 = 2.$ $w_3 = 3.$
$w_4 = 4.$ $w_5 = 1.$

Figure 3.1a is a non-planar rational Bézier curve. 155 samples are taken on the curve to construct the "curvature arc-length" graph in Figure 3.1b. Step 2.1 of our algorithm divides the graph at the peaks. The 3 lines labelled as 'a' in Figure 3.1b cut the graph into 4 regions. Step 2.2 divides each of the four regions into equal area sections, where the area of each section is given as Equation (3.2). The graph is divided into a

total of 18 sections. The 17 subdivision points are also shown on the Bézier curve in Figure 3.1a.

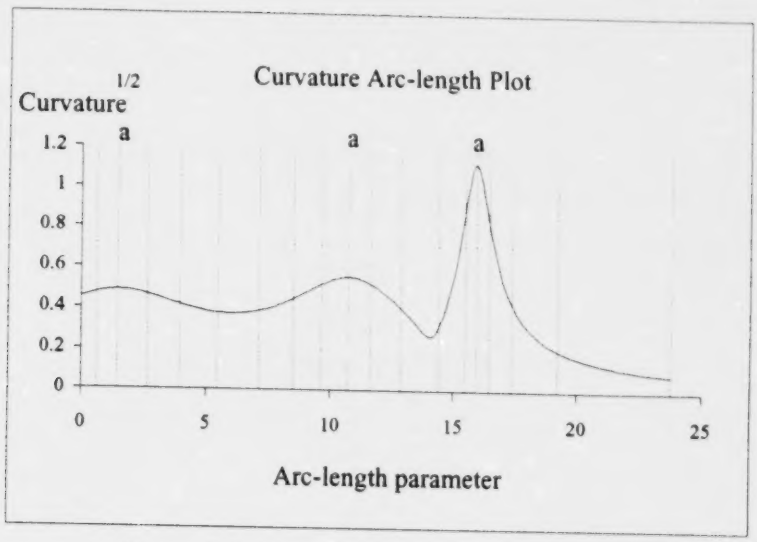


Figure 3.1b: Curvature arc-length graph of Figure 3.1a

3.3 Relation between Chordal Distance and Area under "Curvature Arc-length" Graph

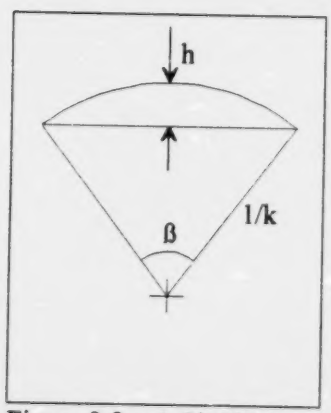


Figure 3.2a: A Single Arc

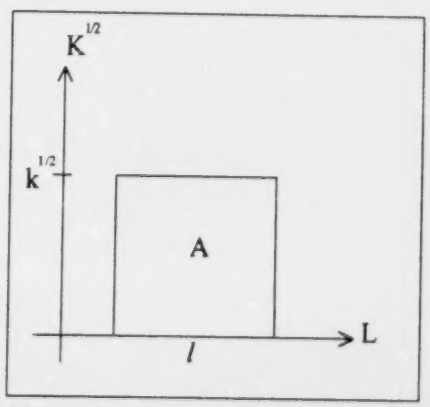


Figure 3.2b: Curvature Arc-length graph

The use of a "curvature arc-length" graph was first inspired by a circular arc. Figure 3.2a shows a circular arc with chordal distance = h, radius = 1/k ,and arc angle = ß. Figure 3.2b is the corresponding "curvature arc-length" graph. The x-axis is the arc-length, and the y-axis is the square root of the curvature. The area of the graph is denoted as A, where

$A = l k^{1/2} = ß / k^{1/2}$.

Given a flat tolerance e, $0<ß<180°$, and $A <= (8e)^{1/2}$, prove $h < e$:

Proof:

The chordal distance, h, in Figure 3.2a can be derived as

$h = (1 - \cos(ß/2)) / k$

$= 2\sin^2(ß/4) / k$

$< ß^2 / (8k)$ because $\sin(ß/4) < ß/4$ for $0<ß<180°$

$= k\, l^2/8$ because $ß = lk$

$= A^2/8$

$<= e$

Therefore, if $(A<=(8e)^{1/2})$ then $h < e$. (3.3)

◊

We have shown that the upper bound of chordal distance of a circular arc can be determined from the area under the "curvature arc-length" graph. We would like to know whether the chordal distance, h, of a Bézier curve segment can be related to the area under the "curvature arc-length" graph. In other words, we would like to investigate the following question:

Given A is the area under the "curvature arc-length" graph of a Bézier curve segment, if $(A=(8e)^{1/2})$, is h<e or h>e?

From our experimental data, we learn that when $(A=(8e)^{1/2})$, h is sometimes greater than e and sometimes less than e. Therefore, our next question is to ask:

Given $(A=(8e)^{1/2})$, can we identify the cases where h>e and h<e?

An answer to the above question is not easy to obtain. However, if a Bézier curve is approximated by a sequence of tiny circular arcs, we can derive two reasonable estimations regarding the relations between A, h, and e.

A Bézier curve can be treated as a sequence of tiny circular arcs where each arc approximates a short range of a Bézier curve. Each arc is so short that the tangents where two adjacent arcs meet can be assumed to be smooth and continuous. In addition, the curvature of the arc is very close to the curvature of the Bézier curve. The "curvature arc-length" graph of the Bézier curve and the arc sequence can be treated as equivalent.

In this section, we will analyze the relation between the chordal distance and the area under the "curvature arc-length" graph based on a sequence of circular arcs.

3.3.1 Chordal Distance of a Circular Arc Sequence

To simplify our analysis in Sections 3.3.1 to 3.3.4, we will make the following assumptions:

1. The circular arc sequence discussed in the following sections has a "curvature arc-length" graph whose area is equal to $(8e)^{1/2}$, where e is the flat tolerance.
2. The tangents where two arcs meet are smooth and continuous.
3. The circular arc sequence lies on a 2 dimensional plane.

4. The circular arc sequence contains no inflexion point.

5. All arc angles are less than 180°.

Let $\mathbf{M}_0$ be a sequence of m circular arcs,

$\mathbf{M}_0 = \{a_{0,j} \mid 0<=j<m\}$,

where $a_{0,j}$ is the jth circular arc,

m is the total number of arcs in $\mathbf{M}_0$,

and $h_{M(0)}$ = chordal distance of $\mathbf{M}_0$.

Figure 3.7a is an example where m=4. Figure 3.7b is the corresponding "curvature arc-length" graph.

Our objective is to answer the following question:

Given A is the area under the "curvature arc-length" graph, and $A=(8e)^{1/2}$,

can we find a way to identify the cases whether $h_{M(0)}>e$ or $h_{M(0)}<e$?

Our answer is the following:

We can identify two specific cases where $h_{M(0)}>e$ or $h_{M(0)}<e$. The identification is based on the shapes of the "curvature arc-length" graph. If the shape of the graph has only one local minimum and no local maximum, or a valley shape, then $h_{M(0)}<e$ is quite plausible. If the shape of the graph has only one local maximum and no local minimum, or a peak shape, then $h_{M(0)}>e$ is quite possible.

We will develop the above answer by linking the relation between $h_{M(0)}$ and e. From Equation (3.3), we know the relation between the chordal distance, h, of a single arc and

the tolerance, e. Therefore, we could deduce a relation between $h_{M(0)}$ and e by finding the relation between $h_{M(0)}$ and h.

To find the relation between $h_{M(0)}$ and h, we reduce the arc sequence $\mathbf{M}_0$ to a single arc by a process called arc reduction. The reduction takes m-1 steps. At each step, a new arc sequence $\mathbf{M}_{i+1}$ is derived from an old arc sequence, $\mathbf{M}_i$, where $0<=i<m-1$. The area under the "curvature arc-length" graph of each sequence is kept constant. The new arc sequence contains 1 fewer arc than the previous sequence. In other words, $\mathbf{M}_i$ is a sequence of (m-i) arcs, and $\mathbf{M}_{i+1}$ is a sequence of (m-i-1) arcs. After the (m-1)th step, $\mathbf{M}_{m-1}$ is an arc sequence that contains only a single arc.

Section 3.3.2 defines the reduction process. Section 3.3.3 derives the relative chordal distance of arc sequences $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ under different conditions. Our final goal in section 3.3.4 is to derive a case where

$$h_{M(0)} < h_{M(1)} < .... < h_{M(m-1)} < e,$$

and a case where

$$h_{M(0)} > h_{M(1)} > .... > h_{M(m-1)} < e.$$

These two cases help us to estimate whether $h_{M(0)}>e$ or $h_{M(0)}<e$.

3.3.2 Arcs Reduction

Let $\mathbf{M}_0$ be the initial arc sequence. Let $\mathbf{M}_i$ be an arc sequence after the ith reduction.

$\mathbf{M}_i$ $= \{a_{i,j} \mid 0<=j<m-i\}$ where $1<=i<=m-1$

$a_{i,j}$ = the jth arc of $\mathbf{M}_i$,

the first subscript denotes the sequence number, the second subscript denotes the arc number,

and $h_{M(i)}$ = chordal distance of $\mathbf{M}_i$.

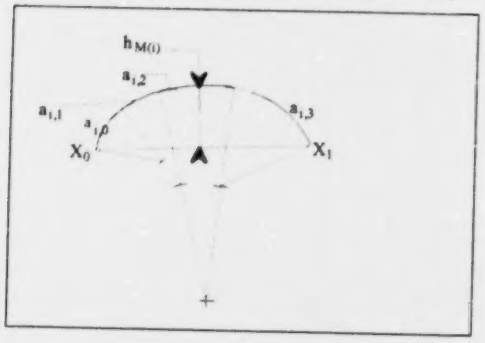


Figure 3.3: Arc sequence $\mathbf{M}_i$, m=4, i=0

Let X_0 and X_1 be the two end points of $\mathbf{M}_i$. The reduction process will replace the two terminal arcs at either end of the arc sequence by a new arc. Let arcs $(a_{i,0}, a_{i,1})$ be the two arcs near point X_0. If the chordal distance, $h_{M(i)}$, is not measured from arc $a_{i,0}$ to line X_0X_1, then the reduction process replaces arcs $(a_{i,0}, a_{i,1})$ with a new arc to form $\mathbf{M}_{i+1}$. If the chordal distance, $h_{M(i)}$, is measured from arc $a_{i,0}$ to line X_0X_1, then the reduction process replaces the other two terminal arcs, $(a_{i,m-i-1}, a_{i,m-i-2})$, near the point X_1 instead. The reduction process is symmetric on both ends of the sequence $\mathbf{M}_i$. For simplicity, we will only give details on how to replace $(a_{i,0}, a_{i,1})$ with a new arc. The replacement of the other two terminal arcs can be derived by simply changing the subscripts of $(a_{i,0}, a_{i,1})$ to the subscripts of $(a_{i,m-i-1}, a_{i,m-i-2})$.

Here are the details of the arc reduction process:

Arcs $(a_{i,0}, a_{i,1})$ in $\mathbf{M}_i$ are replaced by arc $a_{i+1,0}$ in $\mathbf{M}_{i+1}$. All other arcs in $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are identical. Let the curvatures of $a_{i,0}$, $a_{i,1}$ and $a_{i+1,0}$ be denoted by $k_{i,0}$, $k_{i,1}$ and $k_{i+1,0}$; the arc lengths be denoted by $l_{i,0}$, $l_{i,1}$ *and* $l_{i+1,0}$; and the centres of curvature

be denoted by $\mathbf{O}_{i,0}$, $\mathbf{O}_{i,1}$ and $\mathbf{O}_{i+1,0}$ respectively. The curvature, arc length, and center of curvature of arc $a_{i+1,0}$ are defined as follows:

$$k_{i+1,0} = k_{i,1} \tag{3.4}$$

$$l_{i+1,0} = (k_{i,0}^{1/2} l_{i,0} + k_{i,1}^{1/2} l_{i,1}) / k_{i+1,0}^{1/2} \tag{3.5}$$

$$\mathbf{O}_{i+1,0} = \mathbf{O}_{i,1} \tag{3.6}$$

Beside the replaced arcs, all other arcs in $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are the same.

$$\text{arc } a_{i+1,j-1} \equiv \text{arc } a_{i,j} \qquad \text{where } 2<=j<m-i \tag{3.7}$$

The above 4 equations guarantee that the total area in "curvature arc-length" graphs of $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are the same.

3.3.3 Relative Chordal Distances

When $\mathbf{M}_i$ is reduced to $\mathbf{M}_{i+1}$, the chordal distance $h_{M(i)}$ changes to $h_{M(i+1)}$. Let arcs $(a_{i,0},a_{i,1})$ in $\mathbf{M}_i$ be replaced by $a_{i+1,0}$ in $\mathbf{M}_{i+1}$. The relative values of $h_{M(i)}$ and $h_{M(i+1)}$ can be compared in three cases:

1. Curvature of arc $a_{i,0}$ is equal to curvature of arc $a_{i,1}$, $k_{i,0} = k_{i,1}$.
2. Curvature of arc $a_{i,0}$ is less than curvature of arc $a_{i,1}$, $k_{i,0} < k_{i,1}$.
3. Curvature of arc $a_{i,0}$ is greater than curvature of arc $a_{i,1}$, $k_{i,0} > k_{i,1}$.

Case 1: $k_{i,0} = k_{i,1}$

From Equation (3.4),

$$k_{i+1,0} = k_{i,1} = k_{i,0}$$

From Equation (3.5),

$$l_{i+1,0} = (l_{i,0} k_{i,0}^{1/2} + l_{i,1} k_{i,1}^{1/2}) / k_{i+1,0}^{1/2}$$

$$= l_{i,1} + l_{i,0}$$

Therefore, arc $a_{i+1,0}$ is equivalent to the two arcs, ($a_{i,0}$, $a_{i,1}$), joining together. So $h_{M(i)} = h_{M(i+1)}$ when $k_{i,0} = k_{i,1}$.

Case 2: $k_{i,0} < k_{i,1}$

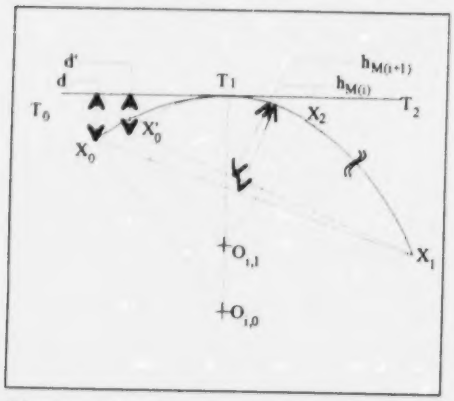


Figure 3.4a: Arc sequences $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$

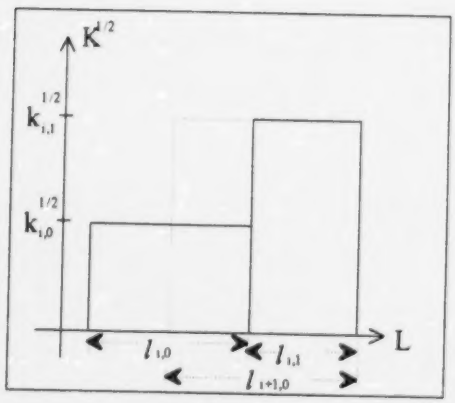


Figure 3.4b

Figure 3.4a shows both $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ arc sequences. Both sequences lie on a 2D plane and contain no inflexion point. $\mathbf{M}_i$ is the arc sequence that spans from point X_0 to point X_1. $\mathbf{M}_{i+1}$ is the arc sequence that spans from point X'_0 to point X_1. Arc X_0 T_1 is arc $a_{i,0}$ in $\mathbf{M}_i$, and arc T_1 X_2 is arc $a_{i,1}$ in $\mathbf{M}_i$. Arc X'_0 X_2 is arc $a_{i+1,0}$ in $\mathbf{M}_{i+1}$. Beside the three arcs $a_{i,0}$, $a_{i,1}$, and $a_{i+1,0}$, the remaining arcs of $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are identical. The remaining arcs are simply drawn as a curve line with no inflexion point in between points X_2 and X_1 in Figure 3.4a. T_1 is the joining point of arcs $a_{i,0}$ and $a_{i,1}$. Line T_0T_2 is the tangent line of arcs $a_{i,0}$ and $a_{i,1}$ at point T_1.

Our objective is to show that $h_{M(i)} > h_{M(i+1)}$. Let d be the perpendicular distance from X_0 to line T_0T_1, and d' be the perpendicular distance from X'_0 to line T_0T_1. The chordal distances of $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are denoted by $h_{M(i)}$ and $h_{M(i+1)}$. $h_{M(i)}$ is measured from

one of the arcs between points T_1 and X_1 to line X_0X_1. If we could show that d>d', then we can conclude that $h_{M(i)}>h_{M(i+1)}$.

$O_{i,0}$ is the curvature centre of arc $a_{i,0}$. From Equation (3.6), $O_{i,1}$ is the curvature centre of arcs $a_{i,1}$ and $a_{i+1,0}$.

Let $ß_{i,0}$ = arc angle of $X_0O_{i,0}T_1$, and

$ß$ = arc angle of $X'_0O_{i,1}T_1$.

Based on assumption (5) in section 3.3.1, $0<ß_{i,0}, ß <=180°$.

Here is the proof of d>d':

From Equation (3.5),

$$l_{i,0}\, k_{i,0}^{1/2} + l_{i,1}\, k_{i,1}^{1/2} = l_{i+1,0}\, k_{i+1,0}^{1/2}$$

$$==> \quad l_{i,0}\, k_{i,0}^{1/2} = (l_{i+1,0} - l_{i,1})\, k_{i,1}^{1/2}$$

because from Equation (3.4), $k_{i+1,0} = k_{i,1}$

$$==> \quad ß_{i,0} / k_{i,0}^{1/2} = ß / k_{i,1}^{1/2}$$

$$==> \quad k_{i,1}^{1/2} / k_{i,0}^{1/2} = (ß/2) / (ß_{i,0}/2) \qquad (3.8)$$

Since $k_{i,0} < k_{i,1}$, therefore $ß_{i,0} < ß$.

Let $a_1 = ß/2$, $a_2 = ß_{i,0}/2$. If $0<ß_{i,0} < ß <=180°$, then $0<a_2 < a_1 <=90°$.

Referring to Figure 3.5, by similar triangles,

$$\sin(ß/2) / \sin(ß_{i,0}/2)$$

$$= \quad \sin(a_1) / \sin(a_2)$$

$$= \quad (a_1 + |q|) / (a_2 + |q|)$$

$$< \quad a_1/a_2$$

$$= \quad ß/ß_{i,0}$$

Therefore, Equation (3.8) becomes

$$k_{i,1}^{1/2} / k_{i,0}^{1/2} > \sin(ß/2) / \sin(ß_{i,0}/2)$$

$$\Longrightarrow \quad 2\sin^2(ß_{i,0}/2) / k_{i,0} > 2\sin^2(ß/2) / k_{i,1}$$

$$\Longrightarrow \quad (1 - \cos(ß_{i,0}) / k_{i,0} > (1 - \cos(ß)) / k_{i,1}$$

$$\Longrightarrow \quad d > d'. \qquad \Diamond$$

Therefore, if $k_{i,0} < k_{i,1}$, then $h_{M(i)} > h_{M(i+1)}$.

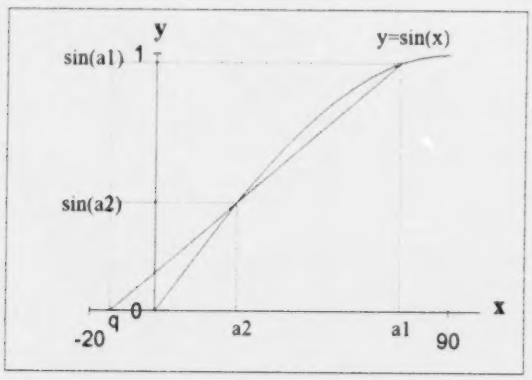


Figure 3.5: Proof of $\sin(a_1)/\sin(a_2) < a_1/a_2$)

Case 3: $k_{i,0} > k_{i,1}$

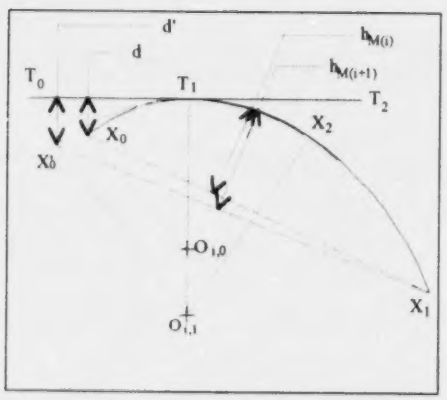


Figure 3.6a: Arc sequences $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$

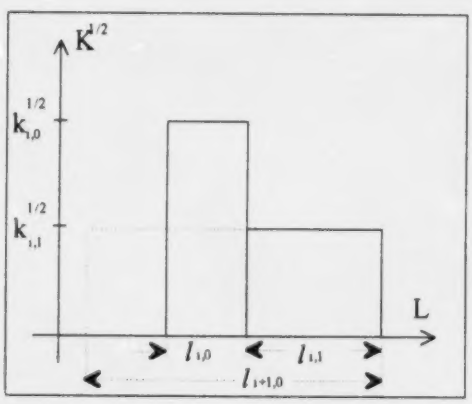


Figure 3.6b

Figure 3.6a is almost a mirror image of Figure 3.4a, where the roles of $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ are flipped. The proof of $h_{M(i)}<h_{M(i+1)}$, is almost the same as case 2. Due to the similarity of the two proofs, we will only state the result here:

If $k_{i,0} > k_{i,1}$, then $h_{M(i)} < h_{M(i+1)}$.

Cases 2 and 3 reductions can be easily recognized graphically on the "curvature arc-length" graphs. Figure 3.4b shows a case 2 graph. The two rectangles in solid line represent the "curvature arc-length" graph of the two arcs $(a_{i,0}, a_{i,1})$ in $\mathbf{M}_i$. In a case 2 reduction, the short rectangle is replaced by a dotted line tall rectangle. The two tall rectangles together represent the new arc $a_{i+1,0}$ in $\mathbf{M}_{i+1}$. Figure 3.6b shows a case 3 graph. The two rectangles in solid line represent the "curvature arc-length" graph the two arcs $(a_{i,0}, a_{i,1})$ in $\mathbf{M}_i$. In case 3 reduction, the tall rectangle is replaced by a dotted line short rectangle. The two short rectangles together represent the new arc $a_{i+1,0}$ in $\mathbf{M}_{i+1}$. These two graphical characteristics help us to identify two types of "curvature arc-length" graph in the next section.

3.3.4 Local Minimum and Local Maximum in Curvature Arc-length Graph

The cases 2 and 3 reductions help us to identify two types of "curvature arc-length" graph. The first type of graph contains one local minimum point and no local maximum point. We refer to this graph as a valley graph. Figure 3.7b is one example. The second type of graph contains one local maximum point and no local minimum point. We refer to this graph as a peak graph. Figure 3.11b is one example.

For a valley graph, we will try to show that the following statement is reasonable: Given an arc sequence, $\mathbf{M}_0$, with a valley "curvature arc-length" graph, where the area under the graph is equal to $(8e)^{1/2}$, then the chordal distance of the arc sequence, $h_{M(0)}$, is likely to be less than e.

The relation between $h_{M(0)}$ and e can be found by reducing the sequence $\mathbf{M}_0$ to a single arc by arc reduction. Let $\mathbf{M}_0$ be a sequence of m arcs. $\mathbf{M}_0$ can be reduced to a single arc, $\mathbf{M}_{m-1}$, through (m-1) steps of arc reduction. If the "curvature arc-length" graph of $\mathbf{M}_0$ is a valley shape, then $\mathbf{M}_0$ is likely to go through more case 3 reductions than case 2 reductions. This is because the reduction process will replace the high curvature arcs on both sides of $\mathbf{M}_0$ with low curvature arcs.

An example in Figures 3.7 to 3.10 may help one to visualize the reduction process. The "curvature arc-length" graph at each step of the reduction process is shown in Figures 3.7b to 3.10b. During the reduction process, a tall rectangle (a high curvature arc) is always replaced by a short rectangle (a low curvature arc). This is the distinct property of case 3 reduction. The reduction process will go through only case 3 reduction if at every step, the chordal distance, $h_{M(i)}$, is always measured at the minimum curvature arc in $\mathbf{M}_i$. If the reduction process goes through only case 3 reduction, then we have the following conclusion:

$$h_{M(0)} < h_{M(1)} < h_{M(2)} < \ldots < h_{M(m-1)} < e.$$

For a generic graph with a valley shape, the reduction process will probably go through more case 3 reductions than case 2 reductions. Therefore, the corresponding chordal distance, $h_{M(0)}$, is plausible to be less than e.

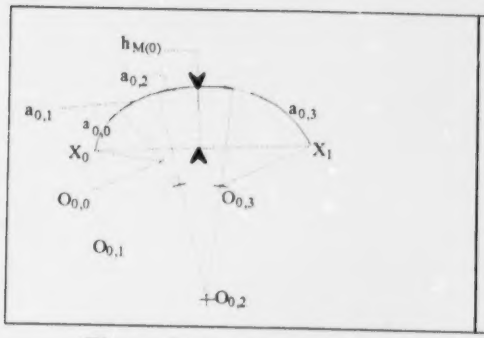


Figure 3.7a: Arc sequence M_0

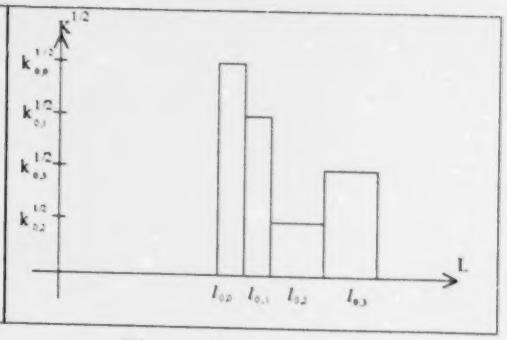


Figure 3.7b

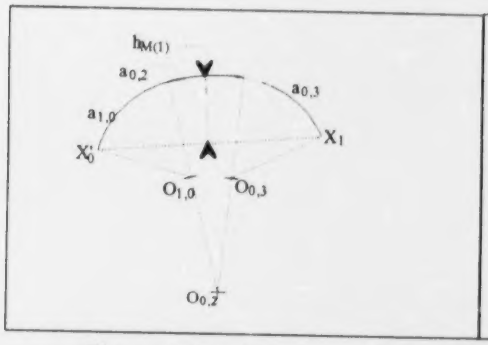


Figure 3.8a: Arc sequence M_1

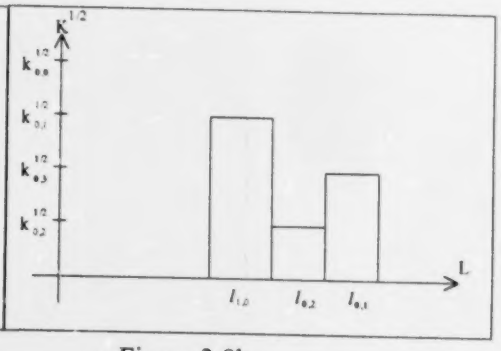


Figure 3.8b

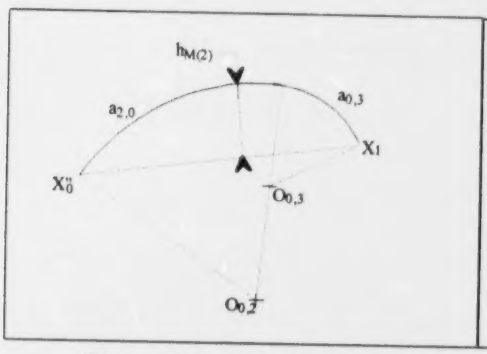


Figure 3.9a: Arc sequence M_2

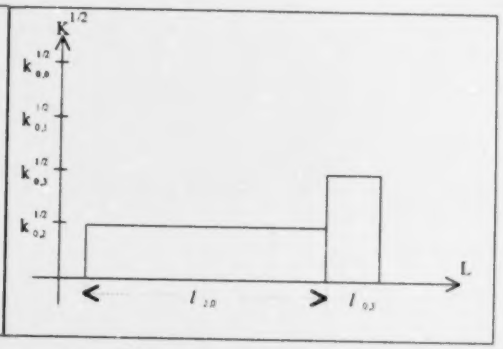


Figure 3.9b

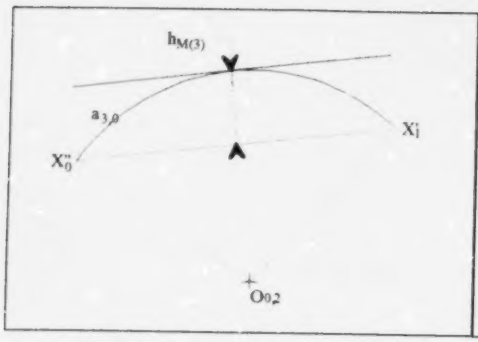


Figure 3.10a: Arc sequence M_3

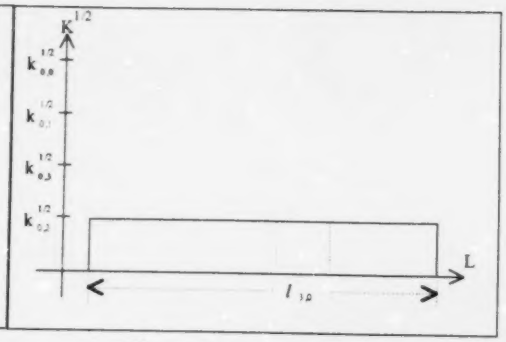


Figure 3.10b

For a peak graph, such as in Figure 3.11b, we will try to show that the following statement is reasonable:

> Given an arc sequence, $\mathbf{M}_0$, with a peak "curvature arc-length" graph, where the area under the graph is equal to $(8e)^{1/2}$, then the chordal distance of the arc sequence, $h_{M(0)}$, is quite possible to exceed e.

Again, the relation between $h_{M(0)}$ and e can be found by reducing the sequence $\mathbf{M}_0$ to a single arc by arc reduction. Let $\mathbf{M}_0$ be a sequence of m arcs. $\mathbf{M}_0$ can be reduced to a single arc, $\mathbf{M}_{m-1}$, through (m-1) steps of arc reduction. If the "curvature arc-length" graph of $\mathbf{M}_0$ is a valley shape, then $\mathbf{M}_0$ is likely to go through more case 2 reductions than case 3 reductions. This is because the reduction process will replace the low curvature arcs on both sides of $\mathbf{M}_0$ with high curvature arcs.

An example in Figures 3.11 to 3.14 may help one to visualize the reduction process. The "curvature arc-length" graph at each step of the reduction process is shown in Figures 3.11b to 3.14b. During the reduction process, a short rectangle (a low curvature arc) is always replaced by a tall rectangle (a high curvature arc). This is the distinct property of case 2 reduction. The reduction process will go through only case 2

reductions if at every step, the chordal distance, $h_{M(i)}$, is always measured at the maximum curvature arc in $\mathbf{M}_i$. If the reduction process goes through only case 2 reduction, then we have the following conclusion:

$$h_{M(0)} > h_{M(1)} > h_{M(2)} > \ldots > h_{M(m-1)} < e.$$

For a generic graph with a peak shape, the reduction process will go through more case 2 reductions than case 3 reductions. Therefore, the corresponding chordal distance, $h_{M(0)}$, is quite possible to exceed e.

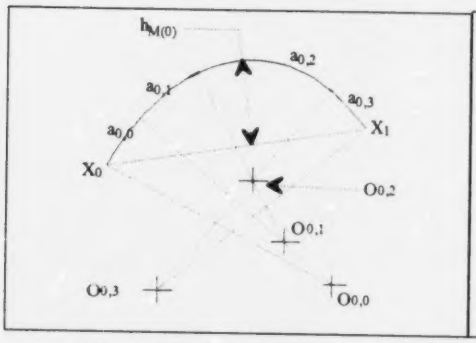


Figure 3.11a: Arc sequence M_0

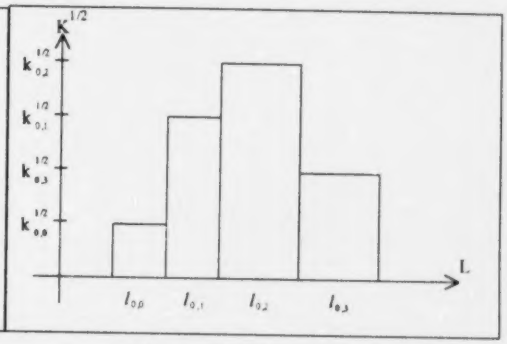


Figure 3.11b

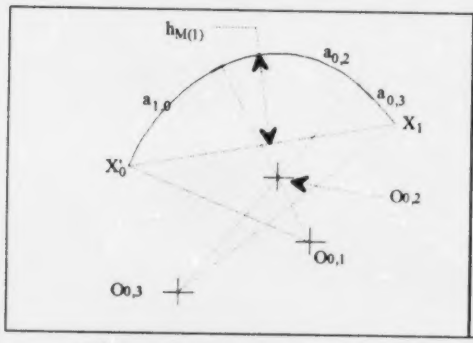


Figure 3.12a: Arc sequence M_1

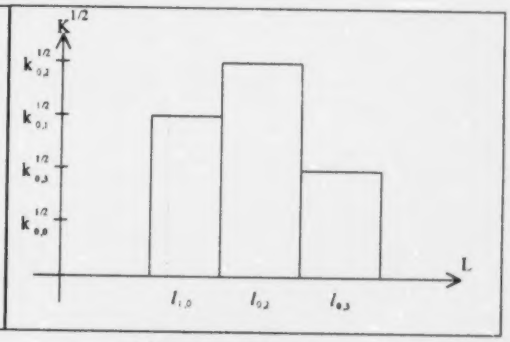


Figure 3.12b

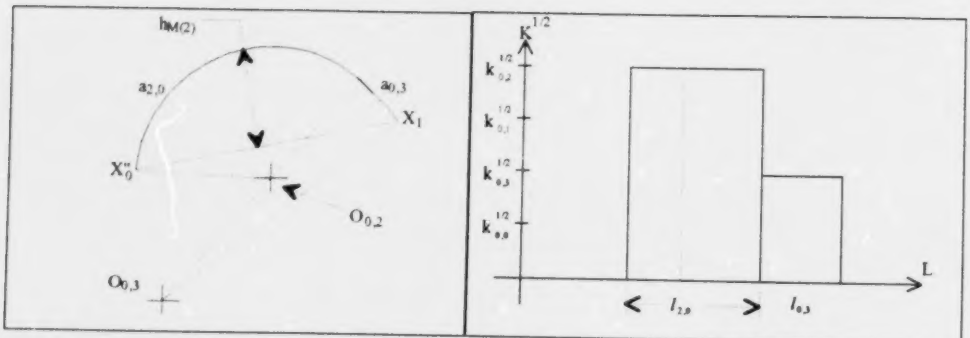


Figure 3.13a: Arc sequence M_2 Figure 3.13b

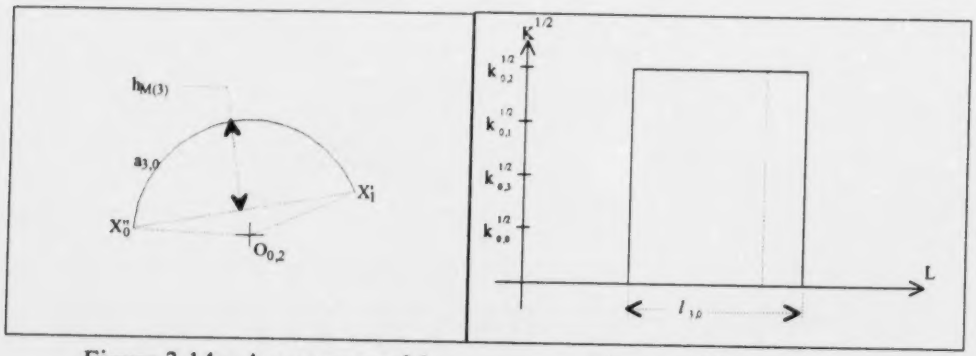


Figure 3.14a: Arc sequence M_3 Figure 3.14b

Based on the above two statements on peak and valley graphs, we design a non-recursive subdivision method that uses the area under the "curvature arc-length" graph as an estimation on the chordal distance. Since a peak shape "curvature arc-length" graph is likely to give an under-estimation on the chordal distance, our algorithm first cuts at every peak in the graph. Each region, between two peaks, is then further cut into equal area sections. Each sectional area is equal to or less than $(8e)^{1/2}$.

The initial design, however, still under-estimates the chordal distances in a small number of cases. On average, 20-90 cases out of every 1000 random Bézier curves contain under-subdivided segments because part of the "curvature arc-length" graph still under-estimates the chordal distances. With a simple modification discussed in section

3.4, our method adequately subdivides a Bézier curve in 99.9% of random cases, and still generates fewer segments than the recursive subdivision methods.

The last remark in this section is on our assumption of 2-D Bézier curve. All the derivation in this section is based on a 2-D Bézier curve approximated by a sequence of circular arcs. If we apply a torsion variable along a 2-D Bézier curve, the 2-D curve becomes a 3-D curve with the same arc-length and curvature. In other words, both curves have the same "curvature arc-length" graph. Assume each subdivided segment of the 2-D curve turns less than 180°. The torsion will twist each 2-D segments into a 3-D helix. The chordal distance of the 2-D segment is usually larger than the chordal distance of the corresponding 3-D helix. A simple example can help one to visualize this: a flat circular arc has a larger chordal distance than the corresponding 3-D helix. Therefore, if the chordal distance estimated from the "curvature arc-length" graph bounds the chordal distance of the 2-D curve, then it also bounds the chordal distance of the corresponding 3-D curve. In Section 3.4, we will test primarily 3-D Bézier curves because 3-D curves is a super set of 2-D curves.

3.4 Empirical Results

Table 3.1: $0.5(8e)^{1/2} <$ Sectional area $A_{si} <= (8e)^{1/2}$

deg	average number of segments			failing cases in our method			favour sampling		favour convex hull	
	our method	sampling	convex hull	cases	h_{ave}	h_{max}	cases	fewer segment	cases	fewer segment
tolerance e=0.2										
2	5.32	6.39	9.05	3.4%	0.2034	0.2270	23.7%	3.9%	2.4%	1.4%
3	7.64	9.25	11.25	4.5%	0.2041	0.2308	16.2%	3.2%	1.0%	0.0%
4	9.38	11.49	13.28	5.3%	0.2054	0.2380	9.4%	2.5%	0.6%	4.5%
5	10.80	13.27	14.69	8.0%	0.2051	0.2273	7.2%	1.8%	0.6%	5.9%
tolerance e=0.1										
2	7.11	9.06	12.61	2.4%	0.1008	0.1018	12.3%	3.6%	0.8%	8.3%
3	10.27	13.03	15.36	5.5%	0.1019	0.1130	4.4%	1.8%	0.3%	0.0%
4	12.64	15.98	18.53	7.2%	0.1030	0.1177	2.0%	1.4%	0.0%	0.0%
5	14.57	18.51	20.61	8.2%	0.1020	0.1190	0.8%	1.0%	0.0%	0.0%
tolerance e=0.05										
2	9.70	12.60	18.02	2.8%	0.0504	0.0521	4.2%	3.4%	0.3%	0.0%
3	14.03	18.46	22.06	3.8%	0.0514	0.0585	0.8%	2.7%	0.0%	0.0%
4	17.25	22.96	26.47	6.7%	0.0513	0.0590	0.3%	3.8%	0.0%	0.0%
5	19.90	26.49	29.17	8.9%	0.0512	0.0566	0.1%	0.0%	0.0%	0.0%
tolerance e=0.02										
2	14.80	20.13	28.13	1.3%	0.0202	0.0204	1.3%	7.8%	0.1%	0.0%
3	21.41	28.80	33.81	6.4%	0.0205	0.0236	0.2%	0.0%	0.0%	0.0%
4	26.39	35.65	41.75	7.2%	0.0205	0.0224	0.0%	0.0%	0.0%	0.0%
5	30.53	41.48	46.16	7.9%	0.0206	0.0238	0.0%	0.0%	0.0%	0.0%
1	2	3	4	5	6	7	8	9	10	11

Table 3.1 is the comparison of three subdivision methods: a non-recursive method using "curvature arc-length" graph; a recursive method using direct sampling as flatness test; and a recursive method using convex hull as flatness test. Three dimensional rational Bézier curves of degree 2 to 5 are tested. Each row of the table represents the

average result of 1000 random curve samples. The control vertices of each sample curve randomly lie within the co-ordinate ranges [0,30]×[0,30]×[0,30]. The weight on the each control vertex randomly lies between 0 and 10. The flat tolerance ranges from 0.2 to 0.02. For comparison purposes, we use the same random seed for the same degree Bézier curves.

The 2nd-4th columns show the average number of segments generated by the three methods. The average number of segment increases as the degree of Bézier curve increases. This is because a high degree curve, in general, has a more complex shape than low degree curve. The average number of segments also increases as the tolerance, e, decreases. This is because smaller tolerance induces more subdivision than larger tolerance on the same curve. Among all cases, our method generates fewer segments on average than the other two methods.

Unlike recursive methods, our method does not test the flatness of a curve segment. Our method estimates the chordal distance of a curve and divides the curve accordingly. To study how accurate our estimation is, we measure the actual chordal distance of all curve segments generated from our method. Again, we use direct sampling to measure the actual chordal distance. We take 10 equally spaced parameter samples on each curve segment and measure the maximum perpendicular distance between the samples and the approximating line. The statistics in column 5 shows that 2%-9% of our subdivided curves contain curve segments that exceed the flat tolerance e. A 9% failing cases means that 90 out of 1000 Bézier curves contain under-subdivided segments.

Among these failing cases, we take the average of all chordal distances that exceed the tolerance e. Column 6 shows this average, denoted as h_{ave}. Column 7 shows the maximum chordal distances, h_{max}, among all the failing segments. It shows that all h_{max}'s lie within 20% of the flat tolerance e. Therefore, even though our method may sometimes under-estimate the chordal distance of a curve segment, the estimation is not too far from the actual chordal distance.

Although our method generates fewer segments than the two recursive methods on average, there are individual cases that the recursive methods generate equal or fewer number of segments than our method. Columns 8 and 10 show the percentage of cases that favour the two recursive methods. They show the percentage of cases where the recursive methods generate equal or fewer segments than our method. The percentage of cases decreases when the degree of Bézier curve increases, and when the tolerance, e, decreases. In other words, on a complex Bézier curve with small tolerance, our method usually performs better than other recursive methods. On a simple Bézier curve with large tolerance, our method may perform more or less the same as other recursive methods.

Among the cases that the recursive methods generate fewer segments than our method, the percentage of fewer segments generated is shown in Columns 9 and 11. As an example, on degree 3 Bézier curve and tolerance e=0.1, there are 44 out of 1000 cases that the recursive method (by sampling) generates equal or fewer segments than our method. Among these 44 cases, the recursive method (by sampling) generates, on average, 1.8% fewer segments than our method. With the same example, only 3 out of 1000 cases that the recursive method (by convex hull) generates equal or fewer segments

than our method. Among these 3 cases, the recursive method (by convex hull) generates the same number of segments as our method.

Returning to the discussion of under-subdividing a Bézier curve in our method, one remedy is to reduce the subdividing sectional area in the "curvature arc-length" graph. Instead of restricting the area as in Equation (3.2), we may restrict the area as

$$0.5(8e)^{1/2} < A_{si} <= c(8e)^{1/2}, \qquad \text{where } .5<c<1.$$

A smaller sectional area, A_{si}, will divide a Bézier curve into smaller segments. Tables 3.2 and 3.3 are the performance statistics based on a smaller sectional area. All cases are generated with the same random seed used in Table 3.1. Table 3.2 restricts the sectional area, A_{si}, between $0.5(8e)^{1/2}$ and $0.95(8e)^{1/2}$. Table 3.3 restricts the sectional area between $0.5(8e)^{1/2}$ and $0.90(8e)^{1/2}$. Columns 5 of Tables 3.2 and 3.3 show significant reduction on the number of failing cases. Table 3.2, in general, has less than 1% of curves that contains under-subdivided segments. Table 3.3 has only 1 case out of a total of 16,000 cases that contains under-subdivided segments.

The trade-off of a smaller sectional area, A_{si}, is more segments. Our method generates, on average, more segments in Table 3.2 than in Table 3.1, and more segments in Table 3.3 than in Table 3.2. Despite the increased number of segments, our method still generates fewer segments on average than both recursive methods. Columns 2 in Tables 3.2 and 3.3 show that our average segment numbers are still smaller than that of the recursive methods.

Columns 8 and 10 show the percentage of individual cases that the recursive methods generate equal or fewer segments than our method. As expected, the percentage of cases favouring recursive methods is larger in Tables 3.2 and 3.3 than in Table 3.1.

As an example, on Table 3.3 of the degree 2 Bézier curve with tolerance e=0.2, there are 46% the cases that the recursive method (by sampling) generates equal or fewer segments than our method. Among these cases, the recursive method (by sampling) generates only 3.9% on average fewer segments than our method. The small percentage difference indicates that the recursive method (by sampling) generates equal number of segments as our method in majority of the 46% cases. With the same example, the percentage of cases that favour recursive method (by convex hull) is only 3.6%, which is still relatively small. In general, our method always performs better than recursive method by convex hull, which is the most common recursive method being used. When compared with recursive method by sampling, our method performs better on a complex Bézier curve with a small tolerance. The recursive method by sampling and our method perform equally well on a simple Bézier curve with a large tolerance.

Table 3.2: $0.5(8e)^{1/2} <$ Sectional Area $A_{si} <= 0.95(8e)^{1/2}$

deg	average number of segments			failing cases in our method			favour sampling		favour convex hull	
	our method	sampling	convex hull	cases	h_{ave}	h_{max}	cases	fewer segment	cases	fewer segment
tolerance e=0.2										
2	5.53	6.39	9.05	0.0%	-	-	32.5%	3.4%	3.1%	1.0%
3	7.94	9.25	11.25	0.1%	0.2151	0.2151	23.3%	3.2%	1.4%	1.1%
4	9.75	11.49	13.28	0.1%	0.2097	0.2097	15.0%	2.8%	1.2%	2.1%
5	11.29	13.27	14.69	0.4%	0.2083	0.2152	12.3%	1.9%	1.3%	2.5%
tolerance e=0.1										
2	7.45	9.06	12.61	0.0%	-	-	18.1%	3.3%	0.8%	12.5%
3	10.76	13.03	15.36	0.5%	0.1028	0.1059	8.7%	1.7%	0.6%	0.0%
4	13.19	15.98	18.53	0.3%	0.1044	0.1045	3.7%	1.0%	0.0%	0.0%
5	15.25	18.51	20.61	0.3%	0.1034	0.1069	1.5%	0.5%	0.0%	0.0%
tolerance e=0.05										
2	10.17	12.60	18.02	0.0%	-	-	6.3%	3.4%	0.4%	0.0%
3	14.68	18.46	22.06	0.2%	0.0519	0.0532	1.8%	2.4%	0.0%	0.0%
4	18.05	22.96	26.47	0.4%	0.0508	0.0514	0.6%	1.4%	0.0%	0.0%
5	20.86	26.49	29.17	0.8%	0.0518	0.0539	0.3%	2.2%	0.0%	0.0%
tolerance e=0.02										
2	15.54	20.13	28.13	0.0%	-	-	2.8%	2.6%	0.1%	0.0%
3	22.52	28.80	33.81	0.0%	-	-	0.6%	2.6%	0.0%	0.0%
4	27.72	35.65	41.75	0.8%	0.0204	0.0209	0.1%	0.0%	0.0%	0.0%
5	32.02	41.48	46.16	0.1%	0.0205	0.0205	0.0%	0.0%	0.0%	0.0%
1	2	3	4	5	6	7	8	9	10	11

Results in Column 3 and 4 are the same as Table 3.1. Both tables use the same random seeds.

Table 3.3: $0.5(8e)^{1/2} <$ Sectional Area $A_{si} <= 0.90(8e)^{1/2}$

deg	average number of segments			failing cases in our method			favour sampling		favour convex hull	
	our method	sampling	convex hull	cases	h_{ave}	h_{max}	cases	fewer segment	cases	fewer segment
tolerance e=0.2										
2	5.80	6.39	9.05	0.0%	-	-	46.4%	3.9%	3.6%	1.7%
3	8.34	9.25	11.25	0.0%	-	-	34.3%	3.9%	2.6%	1.7%
4	10.19	11.49	13.28	0.0%	-	-	25.1%	2.7%	2.0%	1.8%
5	11.83	13.27	14.69	0.0%	-	-	22.8%	2.0%	2.7%	2.2%
tolerance e=0.1										
2	7.83	9.06	12.61	0.0%	-	-	25.4%	4.0%	1.4%	6.7%
3	11.28	13.03	15.36	0.1%	0.1020	0.1020	15.2%	2.0%	0.6%	0.0%
4	13.86	15.98	18.53	0.0%	-	-	9.4%	1.7%	0.2%	0.0%
5	15.99	18.51	20.61	0.0%	-	-	6.7%	0.8%	0.7%	0.0%
tolerance e=0.05										
2	10.69	12.60	18.02	0.0%	-	-	13.0%	1.9%	0.5%	0.0%
3	15.46	18.46	22.06	0.0%	-	-	4.9%	1.6%	0.0%	0.0%
4	18.97	22.96	26.47	0.0%	-	-	2.0%	1.9%	0.0%	0.0%
5	21.96	26.49	29.17	0.0%	-	-	1.5%	0.8%	0.0%	0.0%
tolerance e=0.02										
2	16.35	20.13	28.13	0.0%	-	-	5.3%	2.7%	0.2%	0.0%
3	23.68	28.80	33.81	0.0%	-	-	1.1%	2.1%	0.0%	0.0%
4	29.20	35.65	41.75	0.0%	-	-	0.7%	1.1%	0.0%	0.0%
5	33.73	41.48	46.16	0.0%	-	-	0.3%	0.0%	0.0%	0.0%
1	2	3	4	5	6	7	8	9	10	11

Results in Column 3 and 4 are the same as Table 3.1. Both tables use the same random seeds.

3.5 Conclusion

In this chapter, we present a non-recursive Bézier curve subdivision method. The design of our method is based on the relation between the area under "curvature arc-length" graph and the estimation of the chordal distance of a curve segment. Our method first samples a Bézier curve in small intervals, then constructs the "curvature arc-length" graph, and finally divides the graph into equal area sections. Each sectional area is restricted within the value $(8e)^{1/2}$.

Our method is tested on a large number of random Bézier curves. The results show that it generates fewer segments than the best recursive method. Although our method may under-subdivide a Bézier curve in some cases, the number of these cases can be significantly reduced by setting a smaller sectional area in the "curvature arc-length" graph. Our results show that if the sectional area is restricted between $0.5(8e)^{1/2}$ and $0.9(8e)^{1/2}$, where e is the flat tolerance, then our method adequately subdivides 99.9% of all random Bézier curves. With this restricted area, our method still generates, on average, 20-40% fewer segments than the recursive method by convex hull.

Chapter 4

Bézier Triangle Subdivision

A Bézier triangle is a two dimensional extension of a Bézier curve. Just like a curve, a Bézier triangle can be recursively subdivided and approximated by a mesh of triangular polygons. The traditional recursive subdivision first cuts the triangle into smaller Bézier triangles. Each new triangle is tested for flatness within a tolerance, e. A triangle that is not flat will be subdivided recursively until all resulting triangles are flat.

A simple way to determine the flatness of a triangle involves two testing procedures: curve flatness test and surface flatness test [Fili86a]. A curve flatness test is to test the flatness of the triangle's boundaries. If any of the boundaries fails the curve flatness test, then the triangle as a whole is considered not flat. One of the familiar curve flatness tests is the convex hull method which we discussed in the last chapter.

When all boundaries pass the curve flatness test, then the triangle is subjected to the surface flatness test. A surface flatness test checks the maximum perpendicular distance between the Bézier triangle surface and a planar triangle formed by three control vertices at the corners of the control graph. Again, the usual way of surface flatness test is by convex hull method for Bézier surfaces [Fili86a, Bars87, Lane80b]. This method determines the maximum perpendicular distance between all control vertices and the approximating plane formed by three control vertices at the corners of the control graph. If the maximum distance is smaller than the flat tolerance e, then the Bézier triangle is considered to be flat.

Two testing procedures are used because the curve flatness test is less expensive than the surface flatness test. [Fili86a] empirically finds that when all boundaries of a triangle are flat, then the entire triangle is typically flat or almost flat (requires only 1 or 2 more subdivisions). Therefore, the curve flatness test determines most of the initial triangle subdivisions. Filip's recursive algorithm always chooses to split at the mid-point parameter on the boundaries of the triangle. In this chapter, we apply our "curvature arc-length" graph, described in Chapter 3, to choose a more effective splitting position and our objective is to reduce the number of output planar triangles.

This chapter is organized as follows: Section 4.1 discusses the background of recursive Bézier triangle subdivision; Section 4.2 describes our subdivision method and some implementation issues; and finally, Section 4.3 presents the empirical results.

4.1 Background

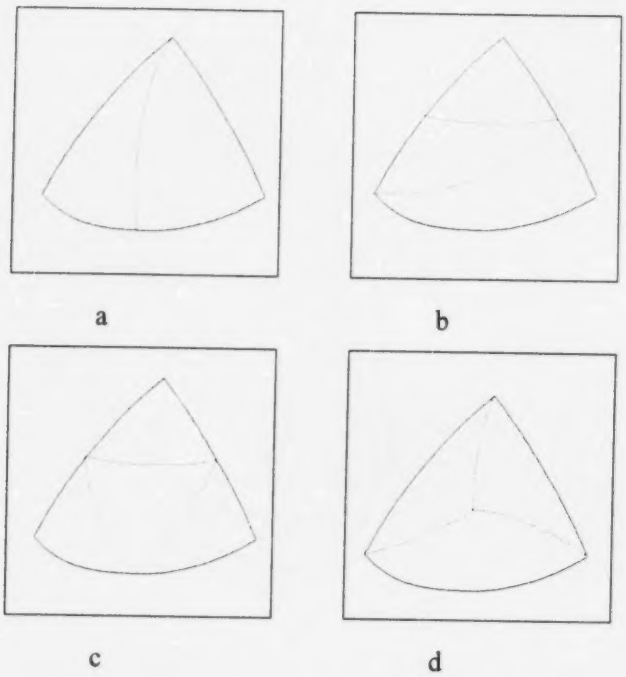


Figure 4.1: Four patterns for subdividing a Bézier triangle

When a Bézier curve is split at each level of a recursive subdivision, it is always split into two Bézier curves. When a Bézier triangle is split at each level, it can be split into two, three, four, or more smaller Bézier triangles at one step. Figure 4.1 shows four of the many possible splitting patterns that can be done at one level. [Gold83] discusses the details on how to perform this subdivision by using degenerate higher dimensional Bézier simplices.

[Fili86a] uses the patterns shown in Figures 4.1a to 1d as the basic patterns in his recursion algorithm. His algorithm can be briefly outlined as follows:

1. Each boundary of a triangle is checked by a curve flatness test. If any of the boundaries fails the curve flatness test, the triangle is split according to pattern 'a', 'b' or 'c'.
2. If all the boundaries are flat, the triangle is checked with surface flatness test. If it fails the second test, it is split according to pattern 'c' or 'd'.
3. If the triangle passes both tests, it is approximated by a planar triangle formed by the three control vertices at the corners of the control graph.
4. Else if the triangle is split into smaller Bézier triangles in step 1 or 2, each new triangle is recursively subdivided by steps 1 to 4.

In his algorithm, the splitting lines of each pattern always cut the boundaries of a Bézier triangle at the mid-point parameters.

[Fili86a] experiments with different combinations of splitting patterns under the following conditions:

1. Use pattern 'b' or 'c' when exactly two boundaries are not flat.
2. Use pattern 'a' or 'c' when exactly one boundary is not flat.
3. Use pattern 'c' or 'd' when all three boundaries are flat, but the surface is not flat.

When none of the boundaries is flat, he uses pattern 'c'.

Let a tuple (p_1,p_2,p_3) be the pattern combination of the above three respective conditions. Then (b,a,d), (c,c,c), and (c,a,d) are the three most significant combinations in [Fili86a]'s results. He concludes that the algorithm using pattern combination (b,a,d) generates the fewest number of triangles, but occasionally gives sliver triangles which are numerically unstable; the algorithm using pattern combination (c,c,c) generates the most regular triangular mesh, but also gives the most number of triangles; and finally, the

algorithm using pattern combination (c,a,d) generates fewer triangles than the second algorithm, but gives more a uniform triangular mesh than the first algorithm. Therefore, he concludes that the third algorithm is the best trade-off for uniformity and the number of triangles. Figure 4.2 is a replica of an example given in his paper. It shows the output triangular mesh from the three algorithms using pattern combinations (c,c,c), (c,a,d), and (b,a,d) respectively.

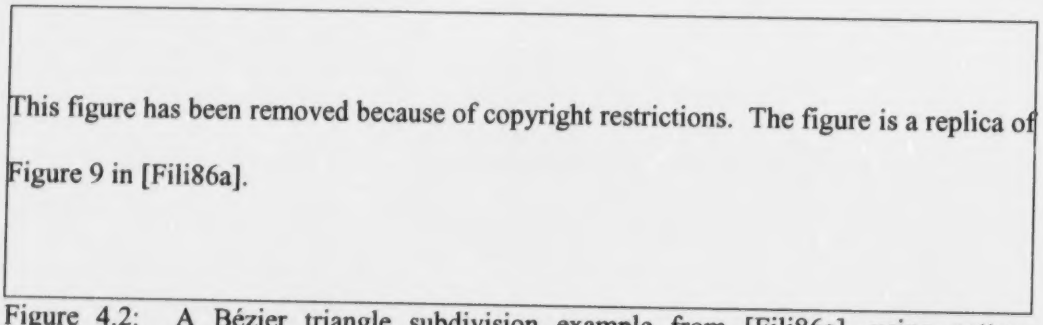


Figure 4.2: A Bézier triangle subdivision example from [Fili86a], using pattern combinations (c,c,c), (c,a,d), and (b,a,d) respectively. The total number of output triangles are 61, 55, and 50.

Figure 4.2a and b shows that there are potential cracks between neighbouring triangles. A crack is generated when two adjacent triangles subdivide at different levels along a flat boundary. Filip solves the crack problem by sticking the subdivision point of a flat boundary onto the approximating line segment, rather than the Bézier triangle surface. This will avoid the appearance of the crack on the screen, but the corner of the approximating triangle may not lie on the Bézier surface. For example, when a Bézier triangle is subdivided by pattern 'c' because it passes all boundary flatness tests but not the surface flatness test, then the middle approximating triangle will have all three corners not lying on the Bézier surface. For some applications in finite element analysis,

this triangular mesh may not be accurate enough. Therefore, we prefer to solve the crack problem by further subdivision. A triangle with 1, 2 and 3 cracks on the boundaries should be subdivided by patterns 'a', 'b', and 'c' respectively. Every subdivided point on the boundaries should lie on the Bézier surface, not on the approximating lines.

With this new crack prevention scheme, Figure 4.2a should be counted as 62 triangles and Figure 4.2b should be counted as 64 triangles. For some general results, we test some random Bézier triangles on the pattern combinations (c,a,d) and (c,c,c) as shown in Tables 4.1a, 4.1b and 4.1c. The objective is to compare the average number of output triangles and the maximum and minimum angles of these triangles. We consider any triangle with maximum angle greater than 140° or minimum angle less than 20° as potential problem triangle. We refer to these triangle as thin triangles.

Table 4.1a:Subdivision using patterns (c,a,d) and (c,c,c).

deg	average number of triangles			average number of thin triangles		
	patterns (c,a,d)	patterns (c,c,c)	percentage difference	patterns (c,a,d)	patterns (c,c,c)	difference (c,a,d) - (c,c,c)
tolerance e =0.4						
2	58.6	68.6	14.5%	6.87	7.10	-0.23
3	72.3	86.0	16.0%	6.60	6.43	0.17
4	94.9	116.9	18.8%	9.54	9.04	0.50
5	114.2	145.8	21.7%	10.71	10.93	-0.22
1	2	3	4	5	6	7

Table 4.1b: Comparison of maximum angles

deg	average number of triangles with max angle > 140°			average angle > 140°			maximum angle > 140°		
	patterns (c,a,d)	patterns (c,c,c)	difference (c,c,c) - (c,a,d)	patterns (c,a,d)	patterns (c,c,c)	difference (c,c,c) - (c,a,d)	patterns (c,a,d)	patterns (c,c,c))	difference (c,c,c) - (c,a,d)
tolerance e =0.4									
2	0.86	0.84	-0.02	148.8°	147.1°	-1.6°	167.8°	160.6°	-7.2°
3	1.11	0.92	-0.19	147.9°	147.1°	-0.8°	167.5°	167.5°	0°
4	1.29	1.09	-0.20	147.2°	147.1°	-0.1°	169.7°	164.5°	-5.1°
5	1.68	1.64	-0.04	147.6°	147.8°	0.1°	168.2°	166.1°	-2.1°
1	2	3	4	5	6	7	8	9	10

Table 4.1c: Comparison of minimum angles

deg	average number of triangles with min angle <20° and max angle < 140°			average angle <20°			minimum angle < 20°		
	patterns (c,a,d)	patterns (c,c,c)	difference (c,a,d) - (c,c,c)	patterns (c,a,d)	patterns (c,c,c)	difference (c,a,d) - (c,c,c)	patterns (c,a,d)	patterns (c,c,c)	difference (c,a,d) - (c,c,c)
tolerance e =0.4									
2	6.01	6.26	0.25	16.02°	16.09°	-0.07°	6.19°	6.93°	-0.73°
3	5.49	5.51	0.02	16.14°	16.18°	-0.04°	6.18°	6.23°	-0.06°
4	8.25	7.95	-0.30	16.00°	16.10°	-0.09°	5.57°	5.62°	-0.06°
5	9.03	9.29	0.26	16.04°	16.25°	-0.22°	4.26°	5.19°	-0.92°
1	2	3	4	5	6	7	8	9	10

Each row in the tables is the result of 500 random Bézier triangles. In order to compare meaningful maximum and minimum triangle angles, we generate Bézier control graphs that do not collapse. We generate the control graph such that the control vertices are ordered in the X and Y dimensions. Each control vertex is randomly located in small rectangular X and Y ranges. Each rectangular XY range of a vertex does not overlap with the XY range of other vertices. The whole graph is located in $[0,30]\times[0,30]\times[0,30]$.

The weight of each control vertex lies between 1 and 10. The flat tolerance is chosen to be 0.4.

Table 4.1a shows the average number of output triangles and thin triangles. Columns 2 to 4 show that pattern combination (c,a,d) generates 14%-22% fewer triangles than pattern combination (c,c,c) on average. Columns 5 to 7 show that pattern combinations (c,a,d) and (c,c,c) generate similar number of thin triangles.

Table 4.1b is the comparison of the thin triangles whose maximum angle is greater than 140°. We refer to these thin triangles as "wide-thin" triangles because of the wide angles. Columns 2-4 are the comparison of the average number of the wide-thin triangles. The combination (c,a,d) on average generates a little more wide-thin triangles than combination (c,c,c). Columns 5-7 are the comparison of the average angles larger than 140°. Columns 8-10 are the comparison of the maximum angles larger than 140°. The combination (c,a,d) usually generates triangles with slightly larger maximum angle (the difference <8°) than the combination (c,c,c).

Table 4.1c is the comparison of the thin triangles that are not accounted for in Table 4.1b. In other words, these are the thin triangles with minimum angle smaller than 20° and maximum angle smaller than 140°. We refer to these thin triangles as "narrow-thin" triangles because of their small minimum angles. Columns 2-4 are the comparison of the average number of the narrow-thin triangles. The combination (c,c,c) on average generates a little more narrow-thin triangles than combination (c,a,d). Columns 5-7 are the comparison of the average angle smaller than 20°. Columns 8-10 are the comparison of the minimum angle smaller than 20°. The combination (c,a,d)

usually generates triangles with slightly smaller minimum angle (the difference<1°) than the combination (c,c,c).

The comparisons show that the differences between maximum angles, minimum angles, and average number of thin angles are very small. We consider the 14%-22% fewer number of triangles is a good trade-off for combination (c,a,d). Therefore, we will use the result from pattern combination (c,a,d) to compare with our results in Section 4.3.

4.2 Bézier Triangle Subdivision

Our Bézier triangle subdivision algorithm is a modification of Filip's recursive subdivision algorithm. Instead of splitting at the mid-point parameter of each non-flat boundary, we split at the optimal subdivision point found by our "curvature arc-length" graph. The "curvature arc-length" graph will find all the necessary subdivision points on the boundary and all the subsequent Bézier triangle subdivisions will cut at these chosen points. The Bézier triangle subdivision pattern will depend on the number of splitting positions on the boundary and the shape of the Bézier triangle.

Each boundary of a triangle is treated as a 3-D Bézier curve. We use the corresponding "curvature arc-length" graph to find all the necessary splitting points on the boundary. We will restrict each sectional area under the "curvature arc-length" graph to be less than $0.90(8e)^{1/2}$. A boundary that contains no necessary splitting point is treated as flat boundary. If a boundary contains more than 1 necessary splitting point, then a point that splits the boundary into two nearly equal lengths is chosen. The other

splitting points are used as the splitting positions of the next level of triangle subdivisions.

We distinguish three shape categories of a Bézier triangle: wide, narrow, and normal shape. A wide triangle is a triangle with a maximum angle greater than 120°. A narrow triangle is a triangle with a minimum angle smaller than 30°. A normal triangle is neither a wide nor a narrow triangle. Figures 4.3a, b and c are the examples of a wide, narrow and normal triangle respectively. Let u, v, w be the three boundary vectors, and let |u|>|v|>|w|. A wide triangle can be determined by

$$\left(\frac{v \cdot w}{|v||w|}\right)^2 > .25$$

A narrow triangle can be determined by

$$\left(\frac{u \cdot v}{|u||v|}\right)^2 > .75$$

a b c

Figure 4.3: Wide, narrow, and normal triangles

The following rules are used to choose the triangle subdivision patterns in Figures 4.1a, ,b, c, and d:

1. If all 3 boundaries should be split by more than 1 points, then choose one splitting point on each boundary and use pattern 'c'.
2. Else if exactly one boundary is flat, then

2.1 if the triangle is a normal triangle, use pattern 'b';

2.2 else if the triangle is a wide or thin triangle,

2.2.1 if the flat boundary is also the shortest boundary, use pattern 'b' (Figures 4.4a and b).

2.2.2 else introduce a point on the flat boundary and use pattern 'c'.

3. Else if exactly two boundaries are flat, then

3.1 if the number of splitting point is equal to 1 and the splitting point was introduced as a crack by Rules 2.2.2, 3.2.3, or 4, then use pattern 'a';

3.2 else,

3.2.1 if the triangle is a normal triangle, use pattern 'a';

3.2.2 else if the triangle is a wide triangle and the non-flat boundary is also the longest boundary, use pattern 'a' (Figure 4.4c);

3.2.3 otherwise, introduce a splitting point on the longer flat boundary and use pattern 'b'.

4. Else if all 3 boundaries are flat and the surface is not flat, introduce a point on the longest flat boundary and split by pattern 'a'.

a b c

Figure 4.4: Rule 2.2.1 for a and b, Rule 3.2.2 for c.

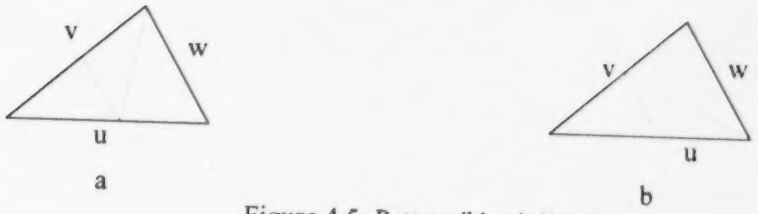


Figure 4.5: Pattern 'b' subdivision

Given one splitting position on two boundaries, there are two possible ways to perform a pattern 'b' subdivision. Figure 4.5 shows the two ways of splitting the triangle with pattern 'b'. Our choice is to first subdivide at the longer boundary, and then subdivide the shorter boundary (Figure 4.5a). This will usually generate more normal triangles than the other alternative.

The splitting points introduced by Rules 2.2.2, 3.2.3, and 4 are potential cracks because they subdivide a flat boundary. The splitting points introduced by using "curvature arc-length" graph are not potential cracks because they are the required subdivision points. For simplicity, we treat potential cracks and non-potential cracks similarly. In other words, if a triangle has one splitting point on each of the three boundaries, then the triangle is subdivided by Rule 1 using pattern 'c' regardless whether these splitting points are potential cracks or not. The only exception is at Rule 3.1. When a potential crack is the only reason to subdivide a triangle, the triangle is subdivided by pattern 'a'. This ensures that a single potential crack will not introduce another potential crack on the other two boundaries of the triangle. If a triangle has one splitting point on two of its boundaries, and both splitting points are potential cracks, then for simplicity, the triangle still follows Rule 2. The worst case in Rule 2 is that two splitting points are replaced by 1 splitting point when Rule 2.2.2 performs a pattern 'c'

subdivision on the two potential cracks. Therefore, a recursive triangle subdivision algorithm based on the above rules is guaranteed to terminate.

Before we describe the algorithm in detail, we should give some details of our data structure. A Bézier triangle is represented by a triangle record.

```
typedef struct    triangle_record {
   double         b[3][DEGREE][DEGREE], w[DEGREE][DEGREE] ;
                  /* DEGREE is the degree of the Bézier triangle */
   int            id ;
   int            *segment[3], segment_size[3] ;
   int            side[3] ;              /* =Left or Right */
   int            surface_flat;          /* =True or False */
   int            shape ;                /* =Wide, Narrow, or Normal */
   double         cos2_angle[3] ;
   int            max_angle, min_angle ; /* =0,1,or 2 */
}        T_RECORD ;
T_RECORD       *Triangle_List[MAX_TRIANGLE] ;
```

The variables 'b' and 'w' store the control graph and the weights of the Bézier triangles respectively. Each triangle has a unique 'id' number, which corresponds to its index position in 'Triangle_List'. Each boundary is represented by an array of indices pointing to the segment record, which will be explained later. The 3 arrays of indices are stored in 'segment[3]'. The sizes of the arrays are stored in 'segment_size[3]'. The variables 'side[3]' is equal to either 'Left' or 'Right' to indicate which side the triangle is on with respect to the boundary segment. This is useful when we have to check the neighbours of the current triangle. An example will be given later. The variable 'surface_flat' is equal to 'True' if the Bézier triangle passes the surface flatness test. A parent triangle that passes the surface flatness test will pass the variable 'surface_flat = True' to its offspring. Therefore, a child that inherits the variable 'surface_flat==True' will not be tested for surface flatness again. The variable 'shape' stores the shape category of the Bézier

triangle. The triangle can either be wide, narrow, or normal shape. The variable 'cos2_angle[3]' stores the cosine squared of the three triangle angles. The indices 'max_angle' and 'min_angle' point to the maximum and minimum angle in 'cos2_angle'. They are determined from the relative distances between the three triangle vertices.

A segment record is defined as follows:

```
typedef struct    segment_record {
    double        u0, u1 ;
    int           neighbour_triangle[2] ;
    int           triangle_boundary[2] ;
                      /* = 0, 1, or 2 */
    double        arc_length ;
    int           crack ;
}  S_RECORD ;
S_RECORD              *Segment_List[MAX_SEG] ;
```

Our curve subdivision algorithm will generate a number of optimal splitting positions along each boundary. These splitting positions are stored as the beginning and ending parameters in an array of segment_record. The variables 'u0' and 'u1' are the beginning and ending parameters of a segment. The variables 'neighbour_triangle[2]' store the id numbers of its two neighbouring triangles. If a segment is part of the outer-most boundary, then one of the 'neighbour_triangle[2]' points to nil. The variables 'triangle_boundary[2]' is equal to 0, 1, or 2. They indicate which boundary of the two neighbouring triangles that the segment is sitting on. The variable 'arc_length' is the approximated arc length of the curve segment. This value is read off from our "curvature arc-length" graph. This value will be useful for choosing a splitting position that splits the boundary into two nearly equal length portions. The variable 'crack' is a flag to indicate that the segment record is introduced by splitting a flat boundary by Rules 2.2.2,

3.2.3, and 4. All segment records generated by our "curvature arc-length" graph will carry 'crack == False'. The segments generated by splitting a flat boundary at rules 2.2.2, 3.2.3, and 4 will carry 'crack == True'.

Here is an example to find the id number of a neighbouring triangle. Let i be a triangle's id. Let B (=0,1,or 2) be one of the boundaries of triangle i. Let m be the number of splitting positions on boundary B. The splitting positions are determined by the "curvature arc-length" graph of boundary B. Boundary B is represented by an array of (m+1) segment_record. The array of indices pointing to segment_record is stored in variable 'segment[B]' of the triangle's record. Let j be the jth segment of boundary B, where 0<=j<=m. The neighbouring triangle of triangle i on the jth segment of the boundary 'B' can be referenced by the following way:

```
T_REC   *tri, *neighbouring_tri ;
int     segment_id, side, n_id ;
#define OPPOSITE(A)     (A==Left?Right:Left)

tri = Triangle_List[i] ;
segment_id = tri->segment[B][j];        /* B = 0, 1, or 2 */
side = tri->side[B] ;
n_id =  Segment_List[segment_id]->neighbour_tri[OPPOSITE(side)] ;
neighbouring_tri = Triangle_List[n_id] ;
```

Our recursive subdivision algorithm can be outlined as follows:

0. Initialize the root Bézier triangle record. First of all, test the flatness of the boundaries by convex hull method. For the non-flat boundaries, find the optimal splitting positions by "curvature arc-length" graph. The optimal subdivision parameters are stored in an array of segment_record. The index to this array is

stored in 'segment[3]' in triangle_record. The size of the arrays are stored in 'segment_size[3]' in triangle_record. A flat boundary is represented by a single segment record with segment_size[i] = 1, where i =0, 1, or 2. The triangle record is pushed into a queue. Set Queue_size=1, Pass=0.

1. If Pass < Queue_size, pull a triangle record from the queue. Queue_size--. Continue with Step 2.

2 If segment_size[0] == segment_size[1] == segment_size[2] == 1 (all boundaries are flat), then

2.1 If surface_flat == False, test for surface flatness and update surface_flat. Continue.

2.2 If surface_flat == True, then

Return the triangle to the end of the queue.

Queue_size++. Pass++. Return to Step 1.

Else continue with Step 2.3.

2.3 Use Rule 4 to split the triangle with pattern 'a'.

2.4 The splitting line in Figure 4.1a is a new boundary of the children triangles. The new boundary is tested for flatness by convex hull method. If the new boundary is not flat, its "curvature arc-length" graph is constructed for finding the optimal splitting positions. The optimal subdivision parameters are stored in an array of segment_record. The index to this array is stored in one of the array 'segment[3]' in triangle_record. The size of the arrays are stored in 'segment_size[3]' in

triangle_record. A flat boundary is represented by a single segment record with segment_size[i] = 1, where i =0, 1, or 2.

2.5 The new segment records are referenced by the two children triangles as an index array in their triangle_records. Each child also inherits part of its parent's boundary information. The parent triangle_record holds all the optimal splitting positions on the parent's boundaries. The children will inherit their corresponding portions of segment_record. The new children records are then pushed to the end of the queue.

Update Queue_size+=2. Set Pass=0. Return to Step 1.

3. Else at least one boundary is not flat,

3.1 Use one of Rules 1, 2, or 3 to subdivide the triangle,

3.2 The splitting lines are the new boundaries of the children triangles. Test the flatness of the new boundaries by convex hull method. Follow similar step in 2.4 to create segment records.

3.3 Update the new segment records and triangle records as in Step 2.5. Push the new children records to the end of the queue.

Update Queue_size according to the number of children. Set Pass=0. Return to Step 1.

The algorithm terminates when (Pass == Queue_size) at Step 1. Queue_size is the size of the queue holding all the subdivided triangles. Queue_size increases whenever a triangle is subdivided into two or more smaller triangles. If a triangle is not subdivided, it is returned to the queue and Queue_size is unchanged. The variable Pass

counts the number of consecutive triangles that do not require subdivision. If any of the triangles is subdivided, 'Pass' is set to 0 and the count starts over again. At every round, Pass is incremented by 1 if segment_size[0,1,2]==1 and surface_flat==True.

A triangle needs to be subdivided if the number of segments on any of the boundaries is greater than 1 or if the triangle does not pass the surface flatness test. The number of segments on the boundaries of a triangle is determined when the boundaries first appear in the root triangle at Step 0, or when the boundaries first appear as the splitting lines of their parent triangles at Step 2.4 or 3.2.

The boundary flatness test is performed at the boundary of the root triangle at the initialization Step 0, and when new boundaries arise as the splitting lines at Steps 2.4 and 3.2. Each boundary is tested for flatness and subjected to "curvature arc-length" procedure only once during initialization (Step 0) or when it first appears as a splitting line (Steps 2.4 and 3.2). No repeated boundary flatness test or "curvature arc-length" procedure is performed. Each boundary is only tested once and all the optimal splitting positions are stored in the segment records. The children triangles will inherit their parent's boundaries segments and inherit the optimal splitting positions of their parent's boundaries.

Each child also inherits the surface flatness information from its parent. If the parent does not pass the surface flatness test, its children inherit "surface_flat==False" and are subjected to the surface flatness test again (Step 2.1). If the parent passes a surface flatness test, all its offsprings will inherit "surface_flat==True" and will not be subjected to the surface flatness test again (Step 2.2). Therefore, our algorithm will keep

the number of surface flatness tests, boundary flatness tests, and "curvature arc-length" procedures to the minimum.

4.2.1 Avoidance of Small Peaks on the Curvature Arc-length Graph

In Chapter 3, we always split the "curvature arc-length" graph at the peak to avoid under-estimating the chordal distance. When a Bézier curve is large in comparison to the flat tolerance, e, then the number of peaks is usually smaller than the number of final subdivision points. However, when a Bézier curve is nearly flat, the number of necessary subdivision points may be smaller than the number of peaks. A Bézier curve that is nearly flat arises very often in the Bézier triangle subdivision because the splitting lines become flatter during the subdivision process. Subdivision at the peak of the "curvature arc-length" graph not only gives more subdivision points than necessary, but also gives uneven subdivision points on the curve. Uneven splitting positions on a boundary of a triangle easily gives a sliver triangle during the Bézier triangle subdivision.

Our remedy is to use a bounding box over the "curvature arc-length" graph. Let n be the number of peaks, k_{max} be the maximum curvature of the graph, and l be the total arc length. The bounding box area is equal to $k_{max}l$. If m<=n is an integer such that

$$k_{max}l <= m(8e)^{1/2},$$

then the simplest way to split the graph is to split at (m-1) equally spaced arc-length parameters. Figure 4.6 is an illustration with n=2, m=2. If the bounding box has an area less than $2(8e)^{1/2}$, then the simplest way is to split at the mid-point of the arc-length. Each half of the bounding box can be thought of as the "curvature arc-length" graph of a

circular arc. Each half of the bounding box determines the maximum chordal distance of the corresponding Bézier curve. If a bounding box area is less than $(8e)^{1/2}$, then the bounding box can be regarded as the "curvature arc-length" graph of a circular arc whose chordal distance is less than e. Any curve with a "curvature arc-length" graph bounded by the bounding box can be derived by flattening the circular arc. Therefore, the derived curve must have a smaller chordal distance than e. The bounding box in the "curvature arc-length" graph determines the maximum chordal distance, e.

If the bounding box area is larger than $n(8e)^{1/2}$, we will use the method in Chapter 3 to find the splitting positions. Although the bounding box method may not give a close estimation on the chordal distance, it gives an upper bound, e, on the chordal distance and is very safe to use.

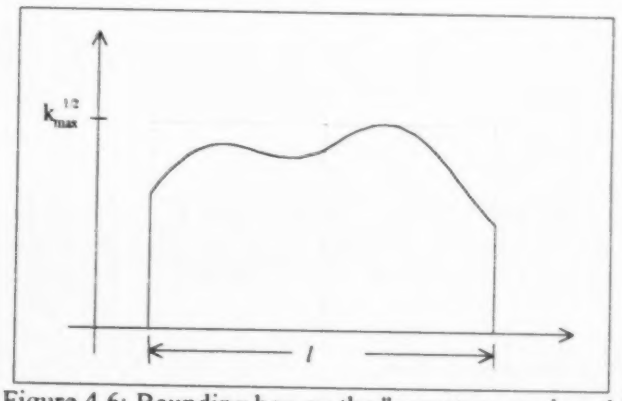


Figure 4.6: Bounding box on the "curvature arc-length" graph

4.2.2 Example of Bézier Triangle Subdivision

Figures 4.7 and 4.8 are the triangular meshes generated from a third degree Bézier triangle by our algorithm and Filip's algorithm (using patterns "c,a,d")

circular arc. Each half of the bounding box determines the maximum chordal distance of the corresponding Bézier curve. If a bounding box area is less than $(8e)^{1/2}$, then the bounding box can be regarded as the "curvature arc-length" graph of a circular arc whose chordal distance is less than e. Any curve with a "curvature arc-length" graph bounded by the bounding box can be derived by flattening the circular arc. Therefore, the derived curve must have a smaller chordal distance than e. The bounding box in the "curvature arc-length" graph determines the maximum chordal distance, e.

If the bounding box area is larger than $n(8e)^{1/2}$, we will use the method in Chapter 3 to find the splitting positions. Although the bounding box method may not give a close estimation on the chordal distance, it gives an upper bound, e, on the chordal distance and is very safe to use.

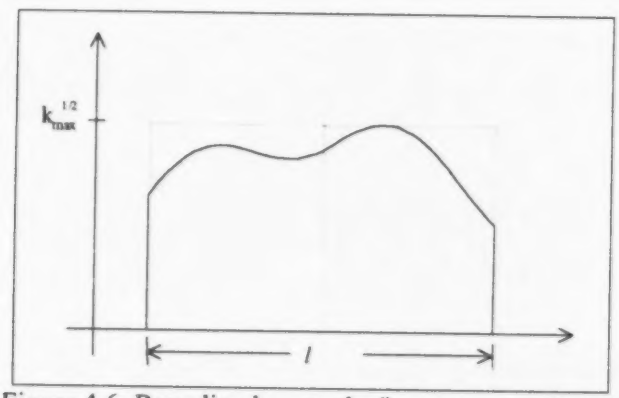


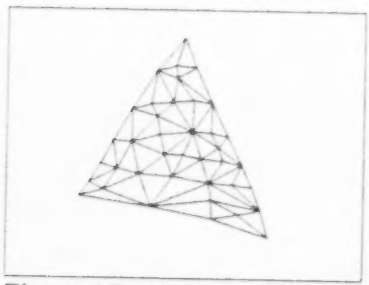
Figure 4.6: Bounding box on the "curvature arc-length" graph

4.2.2 Example of Bézier Triangle Subdivision

Figures 4.7 and 4.8 are the triangular meshes generated from a third degree Bézier triangle by our algorithm and Filip's algorithm (using patterns "c,a,d")

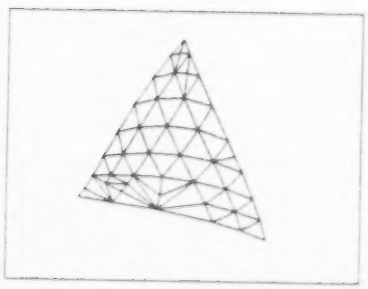
respectively. For a fair comparison, Filip's algorithm is implemented in a way such that any cracks will be further resolved by one of the subdivision patterns 'a', 'b', or 'c'. Therefore, there is no appearance of cracks in either of the triangular meshes.

Both meshes are displayed using a parallel projection on the X-Y plane. Although it appears that our triangular mesh has more thin triangles near the bottom of the mesh than that of Filip's algorithm, this is only a deception because both 3-D meshes are bending away from the view point near the bottom of the mesh. We count the number of thin triangles numerically (max angle >140° or min_angle < 20°) and find that our algorithm actually generates fewer thin triangles than Filip's algorithm in this example. Although Filip's algorithm generates a more regular mesh than our algorithm, our algorithm generates fewer triangles and fewer thin triangles than Filip's algorithm.



Total triangles: 60
Number of triangles
with maximum angle > 140°: 0
with minimum angle < 20° : 2
among these 2 triangles:
minimum angle = 16.3°
average minimum angle = 16.7°

Figure 4.7: Bézier triangle subdivision by our method



Total triangles: 66
Number of triangles
with maximum angle > 140°: 1
maximum angle = 156°
with minimum angle < 20° : 5
among these 5 triangles:
minimum angle = 11.8°
average minimum angle = 15.2°

Figure 4.8: Bézier triangle subdivision by Filip's method, using pattern (c,a,d)

The control vertices of Figures 4.7 and 4.8 are:

$(\mathbf{P}_{0,0}, w_{0,0}) = (5.1, -8.3, -5.8, 4.1)$
$(\mathbf{P}_{0,1}, w_{0,1}) = (0.1, -6., -3.3, 3.)$
$(\mathbf{P}_{0,2}, w_{0,2}) = (3.4, -6.2, -2.5, 4.)$
$(\mathbf{P}_{0,3}, w_{0,3}) = (-6.7, -5.5, -2.7, 5.2)$
$(\mathbf{P}_{1,0}, w_{1,0}) = (4.2, -3.8, 5.9, 3.2)$
$(\mathbf{P}_{1,1}, w_{1,1}) = (-0.6, -2.2, -7.6, 2.)$
$(\mathbf{P}_{1,2}, w_{1,2}) = (-5.4, -4., 4.7, 4.)$
$(\mathbf{P}_{2,0}, w_{2,0}) = (1.3, 0., 5.9, 2.)$
$(\mathbf{P}_{2,1}, w_{2,1}) = (-4.2, 1.2, 2.7, 3.1)$
$(\mathbf{P}_{3,0}, w_{3,0}) = (-0.2, 5.4, -3.2, 6.)$

The flat tolerance used is: $e = 0.2$.

4.3 Empirical Results

Table 4.2a: Rational Bézier triangles subdivision

deg	average number of triangles			average number of thin triangle		
	our method	Filip's method	percentage difference	our method	Filip's method	difference (Filip - our)
tolerance e =0.4						
2	33.6	58.6	42.7%	4.29	6.87	2.58
3	60.0	72.3	17.0%	7.37	6.60	-0.77
4	82.2	94.9	13.4%	9.39	9.54	0.15
5	103.6	114.2	9.2%	13.46	10.71	-2.75
tolerance = 0.2						
2	68.5	112.6	39.2%	7.79	15.13	7.34
3	120.9	138.9	13.0%	12.98	16.65	3.67
4	166.1	193.8	14.3%	18.11	19.45	1.34
5	207.8	232.6	10.6%	26.18	21.43	-4.75
tolerance = 0.1						
2	137.6	230.7	40.3%	13.20	26.17	12.97
3	238.6	275.8	13.5%	22.93	26.94	4.01
4	334.2	376.4	11.2%	33.61	40.99	7.38
5	415.7	447.9	7.2%	47.38	46.29	-1.09
tolerance = 0.05						
2	274.5	445.8	38.4%	25.97	61.54	35.57
3	475.8	535.1	11.1%	43.47	65.43	21.96
4	666.5	759.5	12.2%	61.89	80.42	18.53
5	824.5	900.4	8.4%	87.53	86.35	-1.18
1	2	3	4	5	6	7

Table 4.2b: Maximum angle comparison

deg	average number of triangles with max angle > 140°			average angle > 140°			maximum angle > 140°		
	our method	Filip's method	difference (Filip - our)	our method	Filip's method	difference (Filip - our)	our method	Filip's method	difference (Filip - our)
tolerance e =0.4									
2	0.67	0.86	0.19	148.1°	148.8°	0.7°	168.6°	167.8°	-0.7°
3	0.98	1.11	0.13	146.4°	147.9°	1.5°	162.7°	167.5°	4.8°
4	1.32	1.29	-0.03	147.1°	147.2°	0.1°	170.1°	169.7°	-0.4°
5	2.14	1.68	-0.46	147.4°	147.6°	0.2°	169.4°	168.2°	-1.2°
tolerance = 0.2									
2	0.85	2.31	1.46	147.3°	148.3°	1.1°	169.1°	167.9°	-1.1°
3	1.82	2.39	0.57	147.7°	147.9°	0.1°	172.4°	168.8°	-3.6°
4	2.28	2.90	0.62	147.5°	147.5°	0.0°	172.7°	172.8°	0.1°
5	4.04	3.26	-0.78	148.4°	147.9°	-0.5°	172.2°	169.9°	-2.3°
tolerance = 0.1									
2	1.63	4.36	2.73	147.2°	148.1°	0.9°	173.5°	173.2°	-0.4°
3	2.57	4.67	2.10	147.6°	148.1°	0.5°	168.4°	173.2°	4.8°
4	4.43	5.94	1.51	146.2°	147.7°	1.5°	169.4°	173.1°	3.7°
5	6.68	7.61	0.93	147.7°	147.9°	0.2°	174.2°	171.8°	-2.3°
tolerance = 0.05									
2	2.75	9.62	6.87	147.4°	148.4°	0.9°	167.0°	170.7°	3.7°
3	4.87	10.21	5.34	147.1°	147.3°	0.2°	171.7°	170.4°	-1.3°
4	6.93	11.92	4.99	146.9°	147.5°	0.6°	171.2°	171.1°	0.0°
5	11.34	13.52	2.18	147.5°	147.7°	0.2°	172.5°	173.3°	0.8°
1	2	3	4	5	6	7	8	9	10

Table 4.2c Minimum angle comparison

deg	average number of triangles with min angle <20° and max angle < 140°			average angle <20°			minimum angle < 20°		
	our method	Filip's method	difference (Filip - our)	our method	Filip's method	difference (our - Filip)	our method	Filip's method	difference (our - Filip)
tolerance e =0.4									
2	3.62	6.01	2.39	16.36°	16.02°	0.34°	7.75°	6.19°	1.56°
3	6.39	5.49	-0.90	16.62°	16.14°	0.48°	7.19°	6.18°	1.01°
4	8.07	8.25	0.18	16.71°	16.00°	0.70°	7.62°	5.57°	2.06°
5	11.32	9.03	-2.29	16.67°	16.04°	0.63°	4.92°	4.26°	0.66°
tolerance = 0.2									
2	6.94	12.82	5.88	16.94°	16.14°	0.80°	7.73°	4.80°	2.94°
3	11.16	14.26	3.10	16.90°	16.21°	0.69°	4.82°	4.06°	0.76°
4	15.83	16.55	0.72	16.87°	16.19°	0.68°	6.64°	4.97°	1.67°
5	22.14	18.17	-3.97	16.63°	15.94°	0.69°	4.33°	4.04°	0.29°
tolerance = 0.1									
2	11.57	21.81	10.24	16.94°	16.04°	0.90°	8.32°	4.64°	3.69°
3	20.36	22.27	1.91	16.90°	16.11°	0.80°	7.09°	4.06°	3.03°
4	29.18	35.05	5.87	16.93°	15.85°	1.08°	5.96°	3.99°	1.97°
5	40.70	38.68	-2.02	16.71°	15.85°	0.86°	5.32°	2.19°	3.14°
tolerance = 0.05									
2	23.22	51.92	28.70	17.01°	16.03°	0.98°	6.58°	5.19°	1.40°
3	38.60	55.22	16.62	17.03°	16.20°	0.82°	5.95°	3.18°	2.77°
4	54.96	68.50	13.54	16.98°	16.08°	0.89°	5.61°	3.99°	1.63°
5	76.19	72.83	-3.36	16.92°	15.91°	1.01°	5.84°	2.89°	2.95°
1	2	3	4	5	6	7	8	9	10

Since [Fili86a] shows that there is always a trade-off between the number of output triangles and the regularity of the output triangles, it is only fair to compare the number and quality of the output triangle together. A Bézier triangle subdivision algorithm does not perform better than another algorithm by simply generating fewer triangles. It also has to generate the same quality, or even better quality, triangles than the other algorithm in order to demonstrate its superior performance. Since the bottom

line of the triangle quality is to avoid sliver triangles, we measure the quality by counting the number of the thin triangles that are generated by an algorithm. We define a thin triangle as a triangle with maximum angle larger than 140° or minimum angle smaller than 20°.

Tables 4.2a, 4.2b and 4.2c are the results of comparing our Bézier triangle subdivision algorithm with the algorithm in [Fili86a] using pattern combination (c,a,d). The layout of the tables is very similar to Tables 4.1a, 4.1b, and 4.1c. Table 4.2a compares the average number of output and thin triangles. Table 4.2b compares the average and maximum angles of those thin triangles with maximum angle greater than 140°. Table 4.2c compares the average and minimum angles of those thin triangles that are not accounted for in Table 4.2b. All random Bézier triangles in Tables 4.2 are generated the same way as those in Tables 4.1.

Columns 2 and 3 in Table 4.2a show the average number of output triangles generated by our and Filip's algorithms. Column 4 shows the percentage difference between the two numbers. In general, our method generates 7%-43% fewer triangles than Filip's method. In particular, our method performs significantly better at degree 2 Bézier triangles. The reason is that degree 2 Bézier triangles are not subjected to the surface flatness test. Both our and Filip's algorithms use the same surface flatness test, which measures the displacement distance between the interior control vertices and the approximating triangle formed by the three corners of the Bézier triangle. An interior control vertex is a vertex that does not lie on the boundary of the control graph. Since a degree 2 Bézier triangle does not have interior control vertices, both algorithms skip the

surface flatness test. Our algorithm, which chooses better splitting positions on the boundaries, performs much better than Filip's algorithm.

On the other hand, a Bézier triangle with degree higher than 2 is always subjected to a surface flatness test. A high degree Bézier triangle will have more interior vertices than a low degree Bézier triangle. Therefore, a high degree Bézier triangle has a higher chance of failing the surface flatness test than a low degree Bézier triangle. Better splitting positions on the boundaries may reduce the initial number of triangle subdivisions, but a triangle with 3 flat boundaries also has a high chance of failing the surface flatness test. As a result, our performance in higher degree Bézier triangles is not as prominent as that in lower degree Bézier triangles.

Columns 5 and 6 show the average number of thin output triangles. Columns 5 and 6 are always within 9%-13% of those in columns 2 and 3. The average number of thin triangles increases as the average number of output triangles increases. Column 7 shows the difference between the two averages. A positive value in Column 7 means that our method generates fewer thin triangles than Filip's method. In most cases of degree 2 to 4 Bézier triangles, we generate fewer thin triangles than Filip's method. In the case of degree 5 Bézier triangles, we generate more thin triangles than Filip's method. This is consistent with the results in column 4, in which our performance drops at degree 5 Bézier triangles. For degree 5 Bézier triangles, both methods generate similar number of output triangles and Filip's method generates fewer thin triangles than ours.

The number of thin triangles is further broken down in Table 4.2b and Table 4.2c. Table 4.2b shows the statistics of all thin triangles with maximum angle larger than 140°. We refer to this type of triangle as a 'wide-thin' triangle because of its wide angle.

Columns 2 and 3 show the average number of the 'wide-thin' triangles. Comparing columns 2 and 3 in Table 4.2b with columns 5 and 6 in Table 4.2a, we notice that the number of 'wide-thin' triangles accounts for less than 20% of all the thin triangles. Column 4 in Table 4.2b shows the difference between columns 2 and 3. A positive value in column 4 shows that our method generates fewer 'wide-thin' triangle. In most cases, our method generates fewer 'wide-thin' triangles than Filip's method. The reason is that our method uses pattern 'a' instead of 'd' to subdivide a triangle when all triangle boundaries are flat and the triangle surface is not flat. Filip's method uses pattern 'd' to subdivide non-flat triangle surface, and therefore has higher chance to generate 'wide-thin' triangles than ours.

The average of all angles greater than 140° are shown in columns 5 and 6. The difference of the two average angles is shown in column 7. A positive value in column 7 indicates that our average angle is smaller than that of Filip's. In all cases except one, our average angle is smaller than that of Filip's. Columns 8 and 9 show the maximum angles from each method. Column 10 is the difference between columns 8 and 9. Column 10 shows that both methods generate similar maximum angle in the output triangles.

Table 4.2c shows the statistics of the those thin triangles that have a minimum angle less than 20° and the maximum angle less than 140°. We refer to these triangles as 'narrow-thin' triangles because of its small minimum angle. Columns 2 and 3 show the average number of the 'narrow-thin' triangles. Column 4 shows the difference between columns 2 and 3. A positive value indicates that our method generates fewer 'narrow-thin' triangles than Filip's method. For most cases in degree 2-4 Bézier triangle, our method generates fewer average number of 'narrow-thin' triangles than Filip's

method. For degree 5 Bézier triangle, our method generates more average number of 'narrow-thin' triangles than Filip's method.

Although our method generates on average more 'narrow-thin' triangles than Filip's method for degree 5 Bézier triangle, our 'narrow-thin' triangles have larger average and minimum angles than those from Filip's method. This means that Filip's method is more likely to generate a sliver triangle (triangle with very small angle, such as $<3^{\circ}$) than our method. The main reason is that our method has a different scheme to choose the splitting positions on the boundary. Among all the optimal splitting positions on the boundary, our method chooses the position that will split the boundary into two nearly equal length portions. In many occasions, the best splitting position will still divide the boundary into two unequal length portions. For instance, when there are 2 splitting points evenly distributed on the boundary, the best position will still divide the boundary into 2 uneven portions. Therefore, our triangles are more irregular than Filip's triangles. However, if the boundary's parameterization is extremely uneven and the 0.5 parameter point is shifted to one end, then Filip's method is more likely to generate thin triangles than our method. This is because our method is independent of the parameterization of the boundary. Uneven parameterization is quite common in the random rational Bézier triangles. Therefore, although our method may generate more 'narrow-thin' triangles than Filip's method, our method is less likely to be affected by uneven parameterization and generates fewer extremely small angle triangles.

Since our "curvature arc-length" graph under-estimates the chordal distance of a boundary in less than 0.1% of the cases, we would like to count the number of under-subdivided triangles in our subdivision algorithm. For each output triangle of our

method, we sample 28 equally parameter spaced points and measure the maximum perpendicular distance between the sample points and the approximating triangle. We find that none of the samples in degree 3-5 Bézier triangles exceed the flat tolerance, and less than 0.6% of 500 random cases of degree 2 Bézier triangles contain under-subdivided triangles. This is because the surface flatness test will control strictly the perpendicular distance between the Bézier patch and the approximating plane. Since the degree 2 Bézier triangles skip the surface flatness test, the under-subdivided boundaries become the boundaries of the under-subdivided triangles. Nevertheless, the number of under-subdivided triangles is relatively small in comparison to the total number of triangles. The maximum perpendicular distance that we measured stays within 10% of the flat tolerance.

Based on the results in Tables 4.2a, 4.2b and 4.2c, we recommend the use of our algorithm when the degree of Bézier triangle is smaller than 5. For Bézier triangles of degree larger than or equal to 5, Filip's algorithm using pattern (c,a,d) is a better alternative.

4.4 Conclusion

In this chapter, we present a Bézier triangle subdivision algorithm. Our algorithm is based on [Fili86a]'s recursive algorithm and our "curvature arc-length" graph in Chapter 3. For degree 2 Bézier triangles, our algorithm generates, on average, 38-43% fewer triangles than [Fili86a]'s algorithm. For degree 3 and 4 Bézier triangles, our algorithm generates, on average, 12%-17% fewer triangles than the Filip's algorithm.

For degree 2 to 4 cases, output triangles generated by our algorithm have similar maximum and minimum angles as those generated by Filip's algorithm. For degree 5 Bézier triangles, our algorithm generates 7%-10% fewer triangles than Filip's algorithm. For degree 5 Bézier triangles, our algorithm generates slightly more thin triangles than Filip's algorithm. However, our 'narrow-thin' triangles of degree 2-5, consisting of 80% of all thin triangles, have a larger average and minimum angles than those from Filip's algorithm.

Chapter 5

Concluding Remarks

In this thesis, we present a new refinement procedure for Bézier curves. The new procedure can approximate a Bézier curve with fewer linear segments than the traditional recursive adaptive subdivision. The procedure constructs a "curvature arc-length" graph and divides the graph into equal area sections. Empirically, we find that if the area under each section is restricted to a maximum value of $0.90(8e)^{1/2}$, where e is the flat tolerance, then 99.9% of the random Bézier curves will not contain under-subdivided segments. Moreover, our method generates 20%-40% fewer line segments than the recursive adaptive subdivision (by convex hull flatness test) on average.

The "curvature arc-length" graph is applied to the Bézier triangle subdivision by choosing the optimal splitting positions on the boundaries of a Bézier triangle. In order to avoid long and thin triangles, we design a new scheme for choosing triangle subdivision patterns based on the triangle shapes and the number of flat triangle boundaries. Our experimental results show that our method using "curvature arc-length" graph generates 7%-40% fewer triangles than the traditional recursive Bézier triangle subdivision method (using subdivision patterns "c,a,d"). For Bézier triangles of degree 2 to 4, our method generates fewer thin triangles than the recursive method. For triangles of degree 5, our method generates slightly more thin triangles than the recursive method.

5.1 Future Improvements on Curve Subdivision Algorithm

Although our Bézier curve refinement procedure generates fewer linear segments than the recursive adaptive method, some applications such as interactive graphics display may find our procedure takes too much time. Our refinement procedure computes three square-roots per sample point on the curve: one for the arc-length, one for the curvature, and one for the square-root of the curvature. The following suggested improvements, which require further research, may lead to a faster refinement solution:

5.1.1 Adaptive Bounding Box

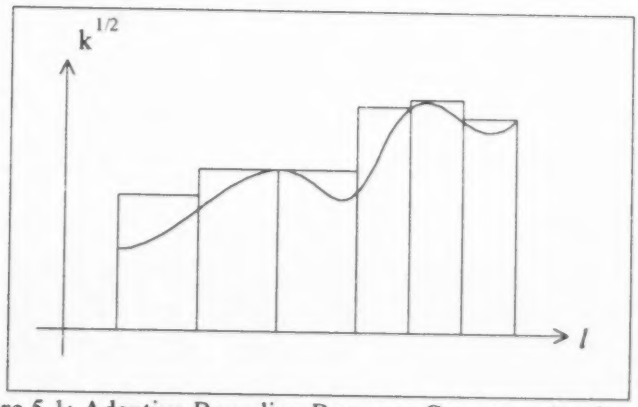


Figure 5.1: Adaptive Bounding Boxes on Curvature Arc-length Graph

In Section 4.2.1, we discuss the use of bounding box to skip the small peaks of the "curvature arc-length" graph. We can also use the same idea to adaptively bound the entire "curvature arc-length" graph by boxes with different sizes (Figure 5.1). If the area of each bounding box is less than or equal to $(8e)^{1/2}$, then the curve segment with the corresponding "curvature arc-length" graph bounded by the box will have a chordal

distance less than e. The argument is the same as given in Section 4.2.1. This method may generate more linear segments than our current method, but it provides the following advantages:

a. We can skip the two square-root operations on the curvature because the bounding box method only finds the local maximum curvature. Once the local maximum curvature is found, we can use the relation $l^4 k_{max}^{\ 2} < (8e)^2$ to check the area of the bounding box. We still need one square-root for the arc-length, l. A good estimate of this square-root can be found by table lookup.

b. We may reduce the number of samples on the curve because we no longer need to evaluate the accurate area under the "curvature arc-length" graph. Fewer sample points will improve the performance.

c. If the bounding box area is less than or equal to $(8e)^{1/2}$, the corresponding curvature segment is guaranteed to stay within the flat tolerance, e. The bounding box on the "curvature arc-length" graph determines an upper bound on the chordal distance.

5.2 2-Dimension Extension of Curvature Arc-length Graph

Finally, we should address the difficulties on the 2-dimension extension of a "curvature arc-length" graph for a Bézier surface. The 2-dimension extension could be a 2-D "curvature arc-length" surface graph, where the z-axis is the square-root of the maximum principal curvature, the x and y axes are corresponding positions on the surface. The idea is to find the relation between the volume under the 2-D surface graph

and the deviation between the Bézier patch and the approximating plane. This is analogous to our 1-D "curvature arc-length" graph where the area under the graph is related to the chordal distance. Two problems have to be solved before we can investigate the 2-D "curvature arc-length" graph:

1. If the volume is used to estimate the displacement between the surface patch and the approximating plane, then the x and y axes of the "curvature arc-length" graph should correspond to the arc-lengths in two perpendicular directions on the surfaces, and the surface area on the xy plane should correspond to the surface area on the Bézier surface. So which pair of parametric directions would give the sample points correctly?
2. A triangulation scheme is needed to subdivide the volume under the "curvature arc-length" graph. The scheme should triangulate the graph on the x-y plane according to the volume under the graph. It should also generate as few triangles as possible.

Bibliography

Bars87.

Barsky, B., T. DeRose, and M. Dippe, *An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-spline Objects*, Report UCB/CSD 87/348, Department of Computer Science, University of California, Berkeley, CA, March 1987.

Bart87.

Bartels, R., J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA, 1987.

Bohm83.

Bohm, W., and G. Farin, "Letter to the Editor on Goldman's algorithm," *Computer-Aided Design*, 15(5):September 1983, 260-261.

Chai74.

Chaikin, G., "An Algorithm for High Speed Curve Generation," *Computer Graphics and Image Processing*, 3(4):December 1974, 346-349.

Chen92.

Cheng, F., "Estimating Subdivision Depths for Rational Curves and Surfaces," *ACM Transaction on Graphics*, 11(2):April 1992, 140-151.

Fari90.

Farin, G., *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide. Second Edition*, Academic Press, Boston, 1990.

Fili86a.

Filip, D., "Adaptive subdivision algorithms for Bézier triangles," *Computer-Aided Design*, 18(2):March 1986, 74-78.

Fili86b.

Filip, D., R. Magedson, and R. Markot, "Surface algorithms using bounds on derivatives," *Computer Aided Geometric Design*, 3(4):December 1986, 295-311.

Fole90.

Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principle and Practice. Second Edition*, Addison-Wesley, Reading, Mass., 1990.

Fors90.

Forsey, D., and V. Klassen, "An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces," in *Proceedings of Graphics Interface '90*, Halifax, Nova Scotia, Canada, May 14-18 1990, 1-8.

Gold83.

Goldman, R., "Subdivision algorithm for Bézier triangles," *Computer-Aided Design*, 15(3):May 1983, 159-166.

Gold87.

Goldman, R., and D. Filip, "Conversion from Bézier rectangles to Bézier triangles," *Computer-Aided Design*, 19(1):January 1987, 25-27.

Gold90.

Goldman, R., "Triangles", *Graphics Gems*, ed. Glassner A., Academic Press, Inc., Boston, 1990.

Gopa91.

Gopalsamy, S., D. Khandekar, and S. Mudur, "A new method of evaluating compact geometric bounds for use in subdivision algorithms," *Computer Aided Geometric Design*, 8(5):November 1991, 337-356.

Lane79.

Lane, J., and L. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing*, 11(3):November 1979, 290-297.

Lane80a.

Lane, J., and R. Riesenfeld, "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):January 1980, 35-46.

Lane80b.

Lane, J., L. Carpenter, T. Whitted, and J. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communications of the ACM*, 23(1):January 1980, 23-34.

Lien87.

Lien, S., M. Shantz, and V. Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces," *SIGGRAPH 87*, 21(4):July 1987, 111-118.

Mort85.

Mortenson, M., *Geometric Modeling*, Wiley, New York, 1985.

Fors90.

Forsey, D., and V. Klassen, "An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces," in *Proceedings of Graphics Interface '90*, Halifax, Nova Scotia, Canada, May 14-18 1990, 1-8.

Gold83.

Goldman, R., "Subdivision algorithm for Bézier triangles," *Computer-Aided Design*, 15(3):May 1983, 159-166.

Gold87.

Goldman, R., and D. Filip, "Conversion from Bézier rectangles to Bézier triangles," *Computer-Aided Design*, 19(1):January 1987, 25-27.

Gold90.

Goldman, R., "Triangles", *Graphics Gems*, ed. Glassner A., Academic Press, Inc., Boston, 1990.

Gopa91.

Gopalsamy, S., D. Khandekar, and S. Mudur, "A new method of evaluating compact geometric bounds for use in subdivision algorithms," *Computer Aided Geometric Design*, 8(5):November 1991, 337-356.

Lane79.

Lane, J., and L. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing*, 11(3):November 1979, 290-297.

Lane80a.

Lane, J., and R. Riesenfeld, "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):January 1980, 35-46.

Lane80b.

Lane, J., L. Carpenter, T. Whitted, and J. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communications of the ACM*, 23(1):January 1980, 23-34.

Lien87.

Lien, S., M. Shantz, and V. Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces," *SIGGRAPH 87*, 21(4):July 1987, 111-118.

Mort85.

Mortenson, M., *Geometric Modeling*, Wiley, New York, 1985.

Patt89.

Patterson, R., and C. Bajaj, *Curvature Adjusted Farameterizations of Curves*, Indiana University - Purdue University at Indianapolis, and Purdue University, July 21, 1989.

Ries75.

RiesenFeld, R., "On Chaikin's Algorithm," *Computer Graphics and Image Processing*, 4(3):September 1975, 304-310.

Sede89.

Sederberg, T., S. White, and A. Zundel, "Fat arcs: A bounding region with cubic convergence," *Computer Aided Geometric Design* 6(3):August 1989, 205-218.

Vlas90.

Vlassopoulos, V., "Adaptive polygonization of parametric surfaces," *Visual Computer*, 6(5):November 1990, 291-298.